Department of Computer Science

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Hamilton, New Zealand

General Boolean Expressions in Publish-Subscribe Systems

A thesis

submitted in partial fulfillment

of the requirements for the degree

of

Doctor of Philosophy

at

The University of Waikato

by

Sven Bittner

May 2008

Abstract

THE INCREASING AMOUNT of electronically available information in society today is undeniable. Examples include the numbers of general web pages, scientific publications, and items in online auctions. From a user's perspective, this trend will lead to information overflow. Moreover, information publishers are compromised by this situation, as users have greater difficulty in identifying useful information.

Publish-subscribe systems can be applied to cope with the reality of information overflow. In these systems, users specify their information interests as subscriptions and, subsequently, only matching information (event messages) is delivered; uninteresting information is filtered out before reaching users. In this dissertation, we consider content-based publish-subscribe systems, a sophisticated example of these systems. They perform the information-filtering task based on the content of provided information. In order to deal with high numbers of subscriptions and frequencies of event messages, publish-subscribe systems are realized as distributed systems. Advertisements—publisher specifications of potential future event messages—are optionally applied in these systems to reduce the internal distribution of subscriptions.

Existing work on content-based publish-subscribe concepts mainly focuses on subscriptions and advertisements as pure conjunctive expressions. Therefore, subscriptions or advertisements using operators other than conjunction need to be canonically converted to disjunctive normal form by these systems. Each conjunctive component is then treated as individual subscription or advertisement. Unfortunately, the size of converted expressions is exponential in the worst case.

In this dissertation, we show that the direct support of general Boolean subscriptions and advertisements improves the time and space efficiency of general-purpose content-based publish-subscribe systems. For this purpose, we develop suitable approaches for the filtering and routing of general Boolean expressions in these systems. Our approaches represent solutions to exactly those components of content-based publish-subscribe systems that currently restrict subscriptions and advertisements to conjunctive expressions.

On the subscription side, we present an effective generic filtering algorithm, and a novel approach to optimize event routing tables, which we call subscription pruning. To support advertisements, we show how to calculate the overlap between subscriptions and advertisements, and introduce the first designated subscription routing optimization, which we refer to as advertisement pruning. We integrate these approaches into our prototype BoP (Boolean publish-subscribe) which allows for the full support of general Boolean expressions in its filtering and routing components.

In the evaluation part of this dissertation, we empirically analyze our prototypical implementation BoP and compare its algorithms to existing conjunctive solutions. We firstly show that our general-purpose Boolean filtering algorithm is more space- and time-efficient than a general-purpose conjunctive filtering algorithm. Secondly, we illustrate the effectiveness of the subscription pruning routing optimization and compare it to the existing covering optimization approach. Finally, we demonstrate the optimization effect of advertisement pruning while maintaining the existing overlapping relationships in the system.

Acknowledgments

I WOULD like to show appreciation to several people who helped me with finishing this dissertation:

I want to thank my supervisors: Annika Hinze, Geoff Holmes, Matt Jones, and Murray Pearson. Naturally, your involvement in the progression of this dissertation took place to different degrees and at different times, and I might have posed different challenges to you. Thanks for your support!

I also would like to thank my fellow PhD students—especially Bryan Genet and Doris Jung for their help and assistance at all times. I am particularly grateful that you two have been around more than only in the lab, specifically you, Doris.

Furthermore, I want to pay thanks to my parents, my brother, and my friends from parts of the world other than New Zealand for providing remote support.

Finally, I appreciate the help of all other people who supported and helped me but who have not been named here, including those of you working behind the scenes.

Thanks for the funding backing this research goes to the Computer Science Department of The University of Waikato for providing a Departmental Scholarship and a Graduate Assistant position. I am also grateful to the New Zealand Government and Education New Zealand for their financial support under the New Zealand International Doctoral Research Scholarships programme.

Contents

List of Figures

List of Tables

Department of Computer Science

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Hamilton, New Zealand

General Boolean Expressions in Publish-Subscribe Systems

A thesis

submitted in partial fulfillment

of the requirements for the degree

of

Doctor of Philosophy

at

The University of Waikato

by

Sven Bittner

May 2008

Abstract

THE INCREASING AMOUNT of electronically available information in society today is undeniable. Examples include the numbers of general web pages, scientific publications, and items in online auctions. From a user's perspective, this trend will lead to information overflow. Moreover, information publishers are compromised by this situation, as users have greater difficulty in identifying useful information.

Publish-subscribe systems can be applied to cope with the reality of information overflow. In these systems, users specify their information interests as subscriptions and, subsequently, only matching information (event messages) is delivered; uninteresting information is filtered out before reaching users. In this dissertation, we consider content-based publish-subscribe systems, a sophisticated example of these systems. They perform the information-filtering task based on the content of provided information. In order to deal with high numbers of subscriptions and frequencies of event messages, publish-subscribe systems are realized as distributed systems. Advertisements—publisher specifications of potential future event messages—are optionally applied in these systems to reduce the internal distribution of subscriptions.

Existing work on content-based publish-subscribe concepts mainly focuses on subscriptions and advertisements as pure conjunctive expressions. Therefore, subscriptions or advertisements using operators other than conjunction need to be canonically converted to disjunctive normal form by these systems. Each conjunctive component is then treated as individual subscription or advertisement. Unfortunately, the size of converted expressions is exponential in the worst case.

In this dissertation, we show that the direct support of general Boolean subscriptions and advertisements improves the time and space efficiency of general-purpose content-based publish-subscribe systems. For this purpose, we develop suitable approaches for the filtering and routing of general Boolean expressions in these systems. Our approaches represent solutions to exactly those components of content-based publish-subscribe systems that currently restrict subscriptions and advertisements to conjunctive expressions.

On the subscription side, we present an effective generic filtering algorithm, and a novel approach to optimize event routing tables, which we call subscription pruning. To support advertisements, we show how to calculate the overlap between subscriptions and advertisements, and introduce the first designated subscription routing optimization, which we refer to as advertisement pruning. We integrate these approaches into our prototype BoP (Boolean publish-subscribe) which allows for the full support of general Boolean expressions in its filtering and routing components.

In the evaluation part of this dissertation, we empirically analyze our prototypical implementation BoP and compare its algorithms to existing conjunctive solutions. We firstly show that our general-purpose Boolean filtering algorithm is more space- and time-efficient than a general-purpose conjunctive filtering algorithm. Secondly, we illustrate the effectiveness of the subscription pruning routing optimization and compare it to the existing covering optimization approach. Finally, we demonstrate the optimization effect of advertisement pruning while maintaining the existing overlapping relationships in the system.

Acknowledgments

I WOULD like to show appreciation to several people who helped me with finishing this dissertation:

I want to thank my supervisors: Annika Hinze, Geoff Holmes, Matt Jones, and Murray Pearson. Naturally, your involvement in the progression of this dissertation took place to different degrees and at different times, and I might have posed different challenges to you. Thanks for your support!

I also would like to thank my fellow PhD students—especially Bryan Genet and Doris Jung for their help and assistance at all times. I am particularly grateful that you two have been around more than only in the lab, specifically you, Doris.

Furthermore, I want to pay thanks to my parents, my brother, and my friends from parts of the world other than New Zealand for providing remote support.

Finally, I appreciate the help of all other people who supported and helped me but who have not been named here, including those of you working behind the scenes.

Thanks for the funding backing this research goes to the Computer Science Department of The University of Waikato for providing a Departmental Scholarship and a Graduate Assistant position. I am also grateful to the New Zealand Government and Education New Zealand for their financial support under the New Zealand International Doctoral Research Scholarships programme.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of electronically available information and content has been increasing significantly over the last few years. To give an example of this trend, Lyman and Varian [LV03] have estimated that new stored information grew about 30 percent a year between 1999 and 2002. When only considering the Internet, examples of information explosion include scientific publications, general web pages, online shopping sites, and online auctions. From a user's perspective, this trend will lead to information overflow. Clearly, both information providers and consumers are compromised by this situation, making it difficult to identify useful information.

Within this dissertation, we consider one kind of system that can be applied by users to cope with the reality of information overflow: *publish-subscribe (pub-sub) systems*. The potential of pub-sub systems to *filter* information in a personalized manner has made them a popular research topic. They fulfill the wish of information providers to selectively disseminate information to interested parties, but they also meet the demands of these parties to filter out uninteresting information. *Content-based pub-sub systems* represent a sophisticated example of such systems, performing the information-filtering task based on the content of provided information. Users can directly state what content they are interested in. Subsequently, only matching information is delivered.

We have found an illustrative example for both the increase in electronically available information and the current popularity of pub-sub research in scientific publications.

We analyzed five major digital libraries (ACM Digital Library[1], Digital

[1]Available at `http://portal.acm.org/dl.cfm`. We considered abstracts in the analysis.

Bibliography & Library Project (DBLP)[2], Google Scholar[3], IEEE Digital Library[4], and SpringerLink[5]) with respect to their content. The overall number of publications per year[6] approximately doubled in the five years from 2001 to 2005, showing the growth in electronically available research output ([Haw01] even identified an exponential growth in scientific articles between 1900 and 2000). In the same time span, the number of publications on pub-sub[7] at least quadrupled, illustrating the increasing proportion of research on this subject. A likely reason for this trend is the potential of pub-sub systems to cope with large amounts of new information and disseminate it to interested parties.

In the following section, we give introductory information about the general pub-sub paradigm to set the context of this dissertation. Having realized that we will not solve all existing problems in the pub-sub area, we have picked one specific issue that needs attention in these systems: canonical conversion of general Boolean user specifications (i.e., subscriptions and advertisements) [Bit06]. We outline this problem in Section 1.2. Then, in Section 1.3 we give an overview of the contributions of this work. We conclude this introductory chapter in Section 1.4 by outlining the structure of the remainder of this document.

1.1 Context: Publish-Subscribe Systems

Pub-sub systems actively deliver incoming information (*event messages*) to their users. Decisions about the delivery of information, that is, what messages are delivered to what users, are based on the descriptions of users. These descriptions of information are referred to as *subscriptions*. Information providers, on the other hand, describe what kind of event messages they will send in the future; these statements are referred to as *advertisements*.

Pub-sub systems can be classified according to the applied information-selection mechanism. This classification into topic-based and content-based systems is based on the level of detail that is available to users to specify

[2]Available at `http://www.informatik.uni-trier.de/~ley/db/`. We considered the title in our analysis.

[3]Available at `http://scholar.google.com/`. We considered the full text in our analysis.

[4]Available at `http://www.computer.org/portal/site/csdl/`. We considered the full text in our analysis.

[5]Available at `http://springerlink.metapress.com/home/main.mpx`. We considered the summary in our analysis.

[6]We derived the total publication number from ACM Digital Library and SpringerLink.

[7]We analyzed all five digital libraries for this content.

subscriptions and to providers to specify advertisements (either using an individual topic or a combination of multiple attributes per specification). In this dissertation, we focus on content-based pub-sub systems, representing the more flexible type of system. In content-based pub-sub systems, the selection of incoming messages is based on their content description; subscriptions thus describe the content users are interested in. Advertisements, conversely, describe what content is potentially sent in the future. The content representation of event messages within this dissertation is assumed to be attribute-value pairs [MFP06], being a popular current representation approach for this type of system. The following example illustrates the concepts in content-based pub-sub systems:

Example 1.1 (Concepts in content-based pub-sub systems) *We use an application area of online auctions within this example and throughout this dissertation.*

Event messages in this application area are information about items on offer. Each event message is represented by the various attributes of a particular item, for example, item description, item condition, current price, current number of bids, and ending time of the auction.

Subscriptions describe the interests of bidders in the auctioning system. For example, buyers would like to be informed about auctions for books of the "Harry Potter" series if the price for a new book copy is below NZ$15.00 one hour before the end of the auction.

Advertisements describe items that will be offered by sellers, acting as information providers in the system. A commercial seller, for example, could state to offer books of the "Harry Potter" series from NZ$15.00 for new book copies and from NZ$10.00 for used book copies.

Content-based pub-sub systems need to be realized as distributed systems in order to scale to large numbers of users, subscriptions, and advertisements, and to support highly frequent event messages. In these systems, subscriptions, advertisements, and event messages are *routed* according to the applied routing algorithm. *Routing optimizations* aim at improving the routing process with respect to efficiency and memory requirements, for example, by exploiting the *covering* among subscriptions, or the *overlap* between subscriptions and advertisements. Within this dissertation, we contribute to different areas of pub-sub research, covering both the central components of these systems and distribution aspects.

There is various research in the pub-sub area, making differing assumptions for typical parameters of these systems. A main assumption of most current approaches is that certain patterns exist among subscriptions. Later, we provide further details about these assumptions on a more technical basis. In this dissertation, we consider *general-purpose* pub-sub systems. With the term "general-purpose", we refer to systems that appropriately fulfill their design goals in range of application settings but are not restricted to exclusively work in "niche" settings.

We restrict our description of pub-sub systems and refrain from introducing further specifics. Chapter 2 provides this information in detail. The descriptions we have given so far, however, allow us to outline a problem in current content-based pub-sub systems in the following section. This problem constitutes the starting point for our research.

1.2 What is the Problem?

The majority of existing work on content-based pub-sub systems focuses on subscriptions and advertisements in conjunctive form. Therefore, subscriptions and advertisements using operators other than conjunction are not directly supported by these systems. However, various applications, in particular more high-level areas such as electronic commerce settings, require subscriptions and advertisements in a general Boolean form (see Chapter 3).

To approach the support of general Boolean subscriptions and advertisements, it is typically argued that systems only need to support conjunctions because any general Boolean expression can be converted to disjunctive normal form. Then, each conjunctive element of such a form can be treated as an individual subscription or advertisement by the system (provided it supports more than one subscription or advertisement per client). At first glance, this argument appears to be sound and was provided, for example, by Mühl and Fiege [MF01], and by Pietzuch [Pie04].

However, on examining the influence of conversion in content-based pub-sub systems more closely, it is questionable whether the conversion approach is a suitable means for these systems. Already one of the fundamental works in the pub-sub area by Yan and García-Molina [YGM94], targeting the *selective dissemination of information* (SDI, as introduced by Salton [Sal68]), addresses the implications of the required conversion when only conjunctions are supported. Yan and García-Molina argue that the handling of general subscriptions as disjunctive normal forms may not be the most efficient processing strategy for subscriptions containing disjunctions [YGM94][8]. However, within their SIFT system [YGM99] they apply the conversion approach and leave the required investigation of the influence of conversion to future work. Research analyzing the effects of conversion has not been undertaken so far, either by Yan and García-Molina or by other researchers.

Instead of investigating the suitability of conversion, the majority of subsequent work in the pub-sub area (i.e., after the seminal work of Yan and García-Molina [YGM94]) has built on their approach without scrutinizing the suitability of converting general Boolean subscriptions and advertisements. As identified previously, various application areas intuitively require disjunctions. For these systems, an investigation of the advantages and disadvantages of supporting general Boolean expressions is even more pressing than for systems targeting the original, pure text-based SDI approach, which allows for the handling of the majority of the existing disjunctions in a specialized way [YGM94]. General content-based pub-sub systems do not offer such an opportunity for handling disjunctions.

The consequences of the conversion approach on pub-sub systems are twofold:

1. Disjunctive normal forms require more memory for storage. They are, in fact, exponential in size in the worst case compared to the original general Boolean form. These memory requirements directly influence the scalability of pub-sub systems (see Section 2.2).

2. The advantageous effect of optimizing algorithms with respect to conjunctions (as done by Yan and García-Molina, and most subsequent work on content-based pub-sub systems) is counterbalanced by the overall increase in the number of subscriptions, and thus the overall increase in the size of the problem to process, after conversion. Even though algorithms might need to compute the result of a common subexpression only once, this result has to be incorporated into all subscriptions and advertisements containing the subexpression.

[8]SDI is one of the historically "original" terms for what evolved into pub-sub systems [Hin03]. Solutions to the filtering problem in the SDI area have been applied to the filtering problem in the content-based pub-sub area [CW03]. The implications and drawbacks of these solutions thus remain in content-based pub-sub systems.

These two effects may not disadvantage systems that perform only a small number of conversions at a given time. For example, database management systems effectively apply the conversion of queries to normal forms [JK84]. The conversion approach is reasonable in these systems because of their pattern of evaluating only few queries simultaneously—queries are transient in these systems. Additionally, database management systems apply query optimization algorithms based on the converted form. Content-based pub-sub systems, however, show the typical pattern of large numbers of subscriptions and advertisements, inherently creating a high system load. These subscriptions and advertisements are stored by the system at all times. Additionally, existing pub-sub approaches for general application settings cannot optimize based on the converted forms as database management systems do.

Hence, the suitability of a conversion approach in these systems is questionable because of the explosion of the already-existing large problem size without the application of an advantageous optimization later on (this being the motivation for conversion in database management systems). We elaborate on the advantages and disadvantages conversion has on the algorithms in content-based pub-sub systems in Section 2.6.

The general topic of conversion to disjunctive normal form and how this influences content-based pub-sub systems recurs throughout this dissertation. Within this work, we will answer the question as to the usefulness of conversion in a step-by-step manner. We outline the contributions of this dissertation in the following section.

1.3 Contributions of this Dissertation

Within this dissertation, we show the advantages of applying content-based pub-sub systems that support general Boolean subscription and advertisement languages. For this purpose, we design solutions that support general Boolean expressions in the required components of pub-sub systems. The main hypothesis of this dissertation consists of two parts and is as follows:

1. *In general-purpose pub-sub systems, a general Boolean filtering approach requires less memory and achieves higher filter efficiency than a conjunctive filtering approach.*

2. *The pruning of filter expressions is an effective routing optimization approach for both general Boolean subscriptions and advertisements.*

(a) *Subscription pruning increases system efficiency and decreases routing table size, independently of the existing covering relationships.*

(b) *Advertisement pruning increases system efficiency and decreases routing table size, while only marginally affecting the existing overlap.*

For this hypothesis, we assume an application scenario requiring more general than pure conjunctive subscriptions and advertisements. We empirically verify the hypothesis throughout this dissertation. Our first step in this process is to provide general-purpose algorithms that support general Boolean subscriptions and advertisements in content-based pub-sub systems. As a second step, we evaluate our general Boolean approaches and compare them to existing conjunctive solutions using an example application scenario that requires general Boolean subscriptions and advertisements.

In detail, the main contributions of this dissertation are:

Application-Scenario Analysis. We investigate the typical requirements of the example application scenario of online auctions. Our contributions include the analysis of event message distributions in online auctions, and the identification of exemplary subscriptions and advertisements in this area. The provision of these details allows for the undertaking of realistic experiments based on a semi-realistic data set[9]. This approach is highly advantageous in comparison to the pure artificial evaluation methods currently employed. It is a first step in the direction of meaningful system analyses based on real-world requirements. Publicizing the results of our application-scenario analysis allows for the repeatability of our experiments as well as for conclusions about their validity.

Support of General Boolean Subscriptions and Advertisements in Local Brokers. We introduce a generic filtering algorithm for general Boolean subscriptions, and symmetrically support advertisements involving general Boolean expressions. The proposed filtering algorithm is a general-purpose solution to the filtering problem but also targets efficient and memory-aware filtering. The algorithm to support advertisements calculates the required overlapping relationships among general Boolean subscriptions and advertisements.

[9]The test data follows the identified specifics of the auctioning area, but is created artificially.

Both solutions are naturally applicable to restricted conjunctive subscriptions and advertisements as well, in that the proposed algorithms inherently support filtering on conjunctive expressions in the same way as general-purpose conjunctive solutions.

Classification Framework and Filtering Algorithm Comparison. We introduce a classification framework for subscriptions. This framework allows for the comparison of the memory requirements of general-purpose filtering algorithms. Using this framework, one can derive which filtering algorithm should be used for a given application scenario (considering the memory requirements), based on the typical attributes of subscriptions. We use this framework to compare the memory usage of our general Boolean filtering algorithm to those of a general-purpose conjunctive solution, and additionally validate these results practically. The framework is analogously applicable to advertisements.

Routing Optimizations. We introduce the first routing optimizations that are practically applicable to general Boolean subscriptions and advertisements. The first optimization is based on subscriptions; the second optimization is based on advertisements. The proposed optimizations naturally work on restricted conjunctive subscriptions and advertisements. Our distinct optimization approach is applicable even if the existing covering optimization shows little potential. Moreover, our subscription-based optimization can be combined with the existing covering approach and can be utilized for the merging of general Boolean subscriptions. The generality of our optimization approach allows us to tailor it to optimize with respect to various target parameters.

Experimental Evaluation. We analyze and evaluate the proposed algorithms for (i) filtering, (ii) calculation of overlap, and (iii) optimizing of the routing process. These analyses include comparisons to general-purpose conjunctive solutions for these three problems in content-based pub-sub systems. The evaluation also involves the analysis of inherent properties of our approaches. Memory requirements and overall time efficiency are evaluated for both routing optimizations. The subscription-based variant additionally includes an analysis of the created network load for event routing; the amount of overlap is analyzed for the advertisement-based variant. The filtering algorithm and the algorithm to calculate the overlap are analyzed with respect to their efficiency properties.

The contributions of this dissertation are broadly reflected within its structure, outlined in the following section.

1.4 Structure of this Dissertation

The structure of this dissertation provides the means to verify our main hypothesis (see Section 1.3, page 6) in a step-by-step manner.

In Chapter 2, we give background information on content-based pub-sub systems, which is required to understand this dissertation. We also analyze existing, related solutions, and identify their assumptions and the implications arising from them. Chapter 3 introduces our application scenario of online auctions that is used as a running example throughout. We analyze the distributions of event messages in this scenario, and identify some typical subscription and advertisement classes.

After these introductory chapters, as a first step to verify our hypothesis, we focus on filtering in the central components of content-based pub-sub systems. A generic filtering algorithm for general Boolean subscriptions is proposed in Chapter 4, alongside general algorithm extensions and improvements. Chapter 5 introduces a characterization framework for general Boolean subscriptions, allowing for the analysis of the memory requirements of filtering algorithms. By applying this framework, we derive conclusions about the behavior of our novel Boolean approach in comparison to a general-purpose conjunctive filtering approach. This chapter concludes with an evaluation of the efficiency properties of the analyzed algorithms.

The next step is to support general Boolean subscriptions in the routing protocols of distributed content-based pub-sub systems. In Chapter 6, we propose subscription pruning, a routing optimization fulfilling this requirement. Subscription pruning approaches the optimization problem from a different perspective than other routing optimizations and allows for optimization based on different target parameters. The different variants of subscription pruning are successively introduced within this chapter. Chapter 7 provides the final milestone on our way to support the general Boolean pub-sub model. We introduce an algorithm to calculate the overlapping relationships between both general Boolean subscriptions and advertisements, and propose an advertisement-based routing optimization, advertisement pruning.

Our last step involves the evaluation of the proposed algorithms. Chapter 8 includes an extensive analysis of empirical experiments. These experiments include (i) the evaluation of the influences of subscription pruning on network load, filter efficiency, and memory requirements, (ii) the comparison of subscription pruning to the existing covering routing optimization, (iii) the evaluation of the efficiency properties of our method to calculate the overlap, (iv) the comparison of this general Boolean method to a conjunctive solution, and (v) the evaluation of the influence of advertisement pruning on memory requirements, efficiency of the overlap calculation, and amount of overlap. We conclude in Chapter 9 by summarizing our results and stating what still needs to be done in the future.

Chapter 2

Background and Related Work

This chapter describes the background and foundations of content-based pub-sub systems that are required to understand this dissertation. Firstly, we describe related work and approaches, presenting the state-of-the-art in the area of content-based pub-sub systems. Providing these facts allows for the classification of our own proposals that are presented later. Secondly, we start to analyze these recent approaches to identify their inherent assumptions and the implications arising out of them. This evaluation substantiates the hypothesis we have proposed in Chapter 1 and intensifies the need to solve the associated research questions.

These two contributions are reflected within the structure of this chapter. We start by introducing generally the notions and concepts of content-based pub-sub systems in Section 2.1. Section 2.2 then elaborates on the accepted quality measures for these systems and identifies the parameters influencing these measures. Current solutions to one of the main tasks in pub-sub systems, the filtering task, are subsequently discussed in Section 2.3. We focus on examining and reviewing the other main task, the routing task, in Section 2.4.

Routing optimizations are one way to optimize pub-sub systems with respect to the identified quality measures; they are the focus of Section 2.5. We conclude this chapter by more thoroughly discussing the influence of canonical conversion of both subscriptions and advertisements in Section 2.6.

2.1 Content-Based Pub-Sub Systems

This section gives a general introduction to content-based pub-sub systems. In the subsequent sections, we refine those concepts that build the main focus of this dissertation. In the following, we introduce content-based pub-sub systems from the familiar viewpoint of database management systems and relate these systems to each other. As we will discover in this section, the correspondences in these systems are not as unambiguous as they might appear, influencing their internal handling of data.

2.1.1 Interaction in Content-Based Pub-Sub Systems

One might look at the concept of content-based pub-sub systems as being similar to database management systems. Although one can argue for this perspective, another viewpoint could contrarily describe content-based pub-sub systems as the opposite of database management systems. We do not want to take sides here because both positions contain legitimate facts, as presented in the following analysis.

Interaction Patterns in Content-Based Pub-Sub Systems

In content-based pub-sub systems, one can find four different interaction patterns. We introduce them in the following paragraphs. Afterwards, we link these concepts to database management systems.

Registering and Deregistering Subscriptions. Similarly to database management systems, content-based pub-sub systems allow their users to define queries. These queries are referred to as *subscriptions* and need to be *registered* with the pub-sub system before their evaluation. Subscriptions are defined with the help of a *subscription definition language* (also referred to as *subscription language*).

The set of all subscriptions is denoted by $\mathcal{S}$, a particular subscription set by $\mathcal{S}_i$ ($\mathcal{S}_i \subseteq \mathcal{S}$), and an individual subscription of this set by $s \in \mathcal{S}_i$. Users, registering such subscriptions, are referred to as *subscribers*, individually denoted by S. The subscriber of a subscription s is abbreviated by $S(s)$.

Subscriptions are valid until they are *deregistered*. In the most general definition, a subscription describes a *Boolean filter expression* (or simply *filter expression*) on event messages. Variables of this expression are called *predicates*, representing simple attribute filters. The concept of event messages is introduced in the following paragraph.

Publishing Event Messages. The incoming information in a pub-sub system is provided by *publishers* in the form of *event messages* (or simply *messages* or *events* within this dissertation). An individual publisher is denoted by P; an event message is abbreviated by e. The publisher of a particular event message e is referred to as $P(e)$.

Generally, event messages are represented by *attribute-value pairs* in content-based pub-sub systems.

Registering and Deregistering Advertisements. Publishers in pub-sub systems have to specify their future event messages and *register* these specifications with the system before sending messages. The term *advertisements* is widely used for the specifications of publishers. They are defined using an *advertisement definition language* (or just *advertisement language*).

The set of all advertisements is denoted by $\mathcal{A}$, a particular set of advertisements by $\mathcal{A}_i$, and an individual advertisement of this set by $a \in \mathcal{A}_i$. Once registered, advertisements need to be *deregistered* to become invalid.

In the most general definition, an advertisement (similarly to a subscription) describes a Boolean filter expression on event messages. An event message e *conforms* to an advertisement a if the filter expression of a evaluates to *true* on e. All event messages sent by a publisher need to conform to one of its registered advertisements. The set of event messages conforming to an advertisement a is denoted by $\mathcal{E}(a)$.

Sending Notifications. The answers to registered subscriptions are provided by the content-based pub-sub system based on the content of the incoming event messages. These answers are called *notifications* and are sent by the system to the respective subscribers. We abbreviate a particular notification by n.

The process of identifying all relevant subscriptions for an incoming message (i.e., those subscriptions whose filter expressions evaluates to *true* for this message) is generally referred to as *filtering* or *event filtering*. For a particular subscription s, the set of all relevant event messages is denoted by $\mathcal{E}(s)$.

When considering a pub-sub system as a black box, this filtering, in combination with the delivery of notifications, is the main task of a pub-sub system: subscribers use the system in order to receive notifications according to their subscriptions. Publishers use the system in order to have their messages delivered to all interested subscribers.

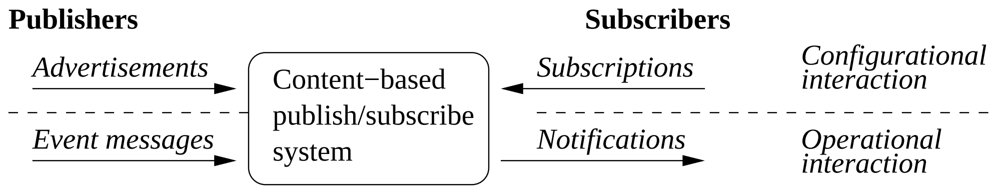


Figure 2.1: Overview of the interaction in pub-sub systems.

These interaction patterns build the means for users to communicate with the help of a pub-sub system. Clearly, the same user might simultaneously act as both subscriber and publisher. That is, the same user might be involved in all interaction patterns, and thus register and deregister subscriptions and advertisements, send event messages, and receive notifications.

We give an overview of the two kinds of users of a content-based pub-sub system and their potential interaction with this system in Figure 2.1. We split the different interaction patterns into two types: the *configurational interaction*, containing the registration of subscriptions and advertisements, and the *operational interaction*, including the publication of messages and the notification about messages.

The content-based pub-sub system is situated between publishers and subscribers and decouples [EFGK03] the communication between these two parties. In the literature, pub-sub systems as decoupling components have thus found variable descriptions, for example, *mediator* [BBC+04, EFGK03, LJ03] and *broker* [BBC+04, HGM01, Leh05].

Correspondence in Database Management Systems

Having introduced the concepts of pub-sub systems, we now relate them to the widely known notions of database management systems. We do so based on two different viewpoints, the *interaction semantics view* and the *data storage view* [BH07].

Interaction Semantics View. Relating the concepts to an interaction semantics view, subscriptions represent database queries and event messages conform to data stored within the database. These correspondences stem from the fact that (i) subscriptions and queries denote user requests that are answered by the respective system, and (ii) event messages and stored data are the basis to provide these answers. This concept of answers, in turn, clearly corresponds to notifications in pub-sub systems and the results to queries in

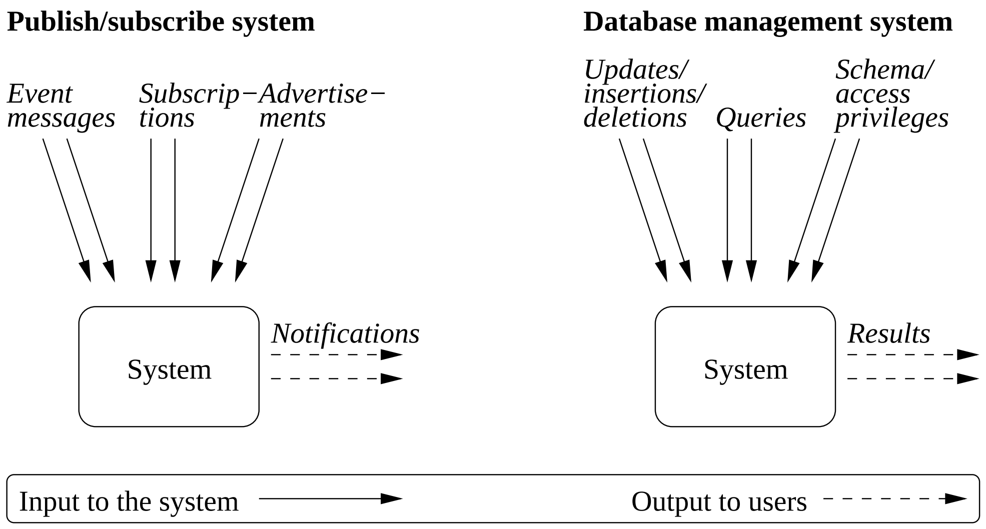


Figure 2.2: Corresponding concepts between pub-sub systems and database management systems when taking an interaction semantics view (corresponding concepts are illustrated at the same position for both kinds of systems).

database management systems.

For advertisements, however, one cannot clearly identify a counterpart in database management systems. Although we cannot find this exact equivalence, the database schema in combination with access privileges to particular tables or table columns can be seen as partially corresponding to advertisements. The advertisements in content-based pub-sub systems, though, are a more general concept. They not only describe the manipulation of a particular type of data (published event messages) but also how the data will be manipulated (the content that will be sent in the future).

We give an overview of these corresponding concepts in content-based pub-sub and database management systems in Figure 2.2. The related notions are arranged in the same positions for both systems to allow for a better overview.

However, there are deep differences between content-based pub-sub systems and database management systems that result from their opposite problem definitions and the implied need to handle data differently. As a consequence, the following observations hold when considering data storage in these systems.

Data Storage View. Subscriptions are long-standing queries that are stored and continuously evaluated by a pub-sub system, until they are finally deregistered and removed from the system. These subscriptions therefore comprise what we call the *subscription base*. Because this subscription base needs to

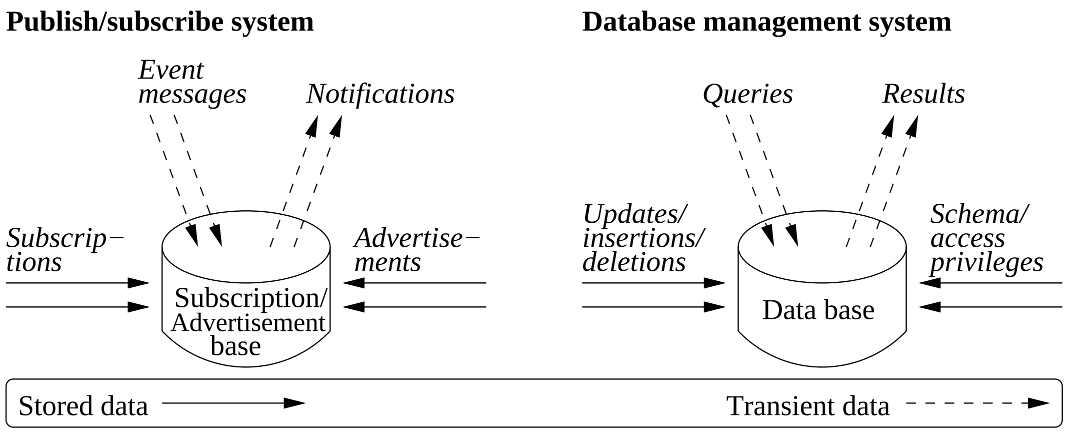


Figure 2.3: Corresponding concepts between pub-sub systems and database management systems when considering the data storage (corresponding concepts are illustrated at the same position for both kinds of systems).

be stored within the system, it is the counterpart to the data that is stored in database management systems, the *data base*. Hence, these two concepts build a component that is known to the respective system.

Database queries, however, are not known in advance. They are sent to the database management system once, are subsequently executed, and finally the system returns the results to the issuer of the query. Similarly, this attribute holds for the individual event messages from publishers. At the (highly frequent) occurrence of event messages, the pub-sub system needs to find all relevant subscriptions in its subscription base. Thus, database queries and event messages are corresponding concepts in these systems.

The correspondence of the remaining concepts aligns with the findings when taking the interaction semantics view. Notifications conform to query results, and advertisements partially match with database schema and access privileges. The latter needs to be stored by the system. We refer to stored advertisements as the *advertisement base* in the following.

We illustrate these corresponding concepts in Figure 2.3. Again, the related notions are arranged in the same positions for both systems. Out of this figure, one can clearly identify the opposite problem definition of content-based pub-sub and database management systems. One system (database management system) answers queries based on stored data; for the other system (pub-sub system) incoming data (event messages) leads to answers (notifications) to stored queries (subscriptions).

This opposite problem definition in content-based pub-sub and database management systems strongly influences the internal handling of stored data, that is, subscriptions and advertisements. We further elaborate on these effects and their implications for the design of content-based pub-sub systems in Section 2.6.

2.1.2 Architecture of Content-Based Pub-Sub Systems

Having presented the interaction patterns in content-based pub-sub systems, we now explain their basic architecture.

One of the main quality measures in pub-sub systems is scalability. Large-scale content-based pub-sub systems thus require a distributed implementation [Müh02] to handle large numbers of clients, as well as subscriptions, advertisements, and event messages. These distributed pub-sub systems consist of a network of so-called *brokers*, individually referred to as B. Subscribers S and publishers P connect to and interact with one of these brokers which is then referred to as their *local broker* $B(S)$ and $B(P)$, respectively. The sets of subscribers and publishers connected to broker B are denoted by *local subscribers* $S(B)$ and *local publishers* $P(B)$, respectively. We use the term *client* to refer to either subscribers or publishers.

Each broker of the system is assigned to a particular set of publishers and subscribers, and offers the pub-sub functionality and interaction patterns (see Section 2.1.1) to these clients. In doing so, the broker hides the distributed nature of the pub-sub system and transparently performs the required actions within the distributed system.

The internal functioning of a distributed content-based pub-sub system strongly depends on the network topology which is used. Most current research prototypes targeting efficiency aspects (another important quality measure) assume the brokers to be connected by a fixed acyclic (overlay) graph structure (or a fixed spanning tree structure), for example, A-mediAS [Hin03], Gryphon [BCM+99], Jedi [CNF01], Kyra [CS04], Padres [LHJ05], Rebeca [Müh02], Siena [CRW01], and XRoute [CF03]. The CBCB scheme [CRW04] supports several spanning trees.

In this dissertation, we build on the foundations laid by these systems and assume an acyclic overlay network as the topology of the broker network. We also assume stable brokers and clients, and error-free connections, both broker-broker and broker-client connections. We give an overview of our system view in Figure 2.4. It contains a set of seven brokers, B_1 to B_7, that are connected

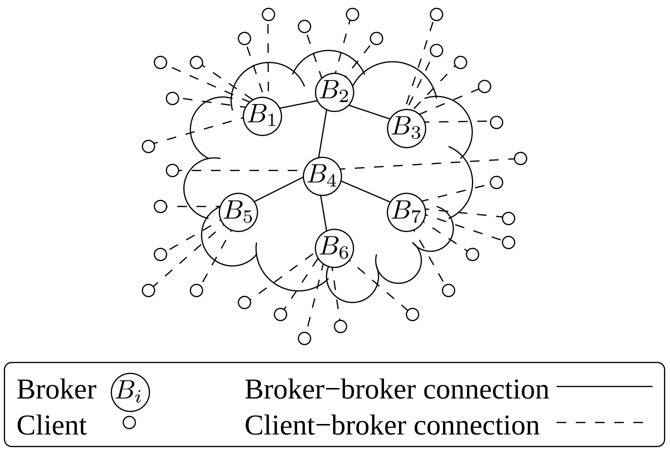


Figure 2.4: Overview of a distributed pub-sub system with seven brokers and several clients.

acyclically. Each of these brokers is connected to several clients: their local clients.

In the literature, one can find extensions to this basic scheme, for example, to allow for dynamic reconfiguration of the network, including broker components as well as broker-broker connections [CFMP04]. We do not specifically target such aspects in this dissertation; they clearly go beyond the scope of our work. Rather we concentrate on the core functionality of content-based pub-sub systems and the tasks created thereby, as described in the following subsection. Nevertheless, the work in this dissertation is not restricted to an assumed architectural scheme.

2.1.3 Tasks in Content-Based Pub-Sub Systems

In content-based pub-sub systems, one needs to solve a large range of tasks. Within this dissertation, we mainly focus on three fundamental tasks, namely the event filtering, the event routing, and the overlapping task, that provide the means to support the interaction patterns we introduced in Section 2.1.1. We define these tasks in the following three subsections. For completeness, we additionally sketch other extended tasks, which are outside the focus of this dissertation.

Event Filtering Task

As already introduced in the last section, the event filtering task is one of the main tasks in content-based pub-sub systems. We formulate it as follows:

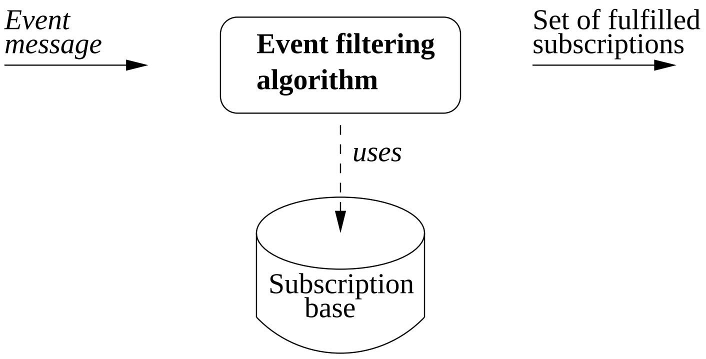


Figure 2.5: Illustration of the task of an event filtering algorithm: based on the subscription base, an incoming event message leads to a set of fulfilled subscriptions.

Definition 2.1 (Event Filtering Task) *A broker of the system has been given a set of subscriptions $\mathcal{S}_i$. For each incoming event message e, it needs to find every subscription $s \in \mathcal{S}_i$ whose filter expression evaluates to* true *on e.*

Within this dissertation, we use the following naming conventions. If the filter expression of s evaluates to *true* on e, we say subscription s is *fulfilled* by event e, which means that event message e *matches* subscription s. An algorithm that solves the filtering task is denoted by the *event filtering algorithm* or just *filtering algorithm*. We illustrate the task of an event filtering algorithm in Figure 2.5.

Within this dissertation, we develop a novel filtering algorithm (Chapter 4). We give an introduction to current filtering algorithms in Section 2.3.

Event Routing Task

Distributed pub-sub systems need to solve an additional crucial task in addition to event filtering: the *event routing task* (or simply *routing task*). We define it as follows:

Definition 2.2 (Event Routing Task) *The system has been given an incoming event message e and a set of brokers $\mathcal{B}$. It needs to determine all brokers $B \in \mathcal{B}$ that have connected local subscribers with registered subscriptions that are fulfilled by e.*

We refer to an algorithm solving the event routing task as the *event routing algorithm*, or simply *routing algorithm*. Note that this algorithm can be realized either distributed or centralized. Within this dissertation, we do not

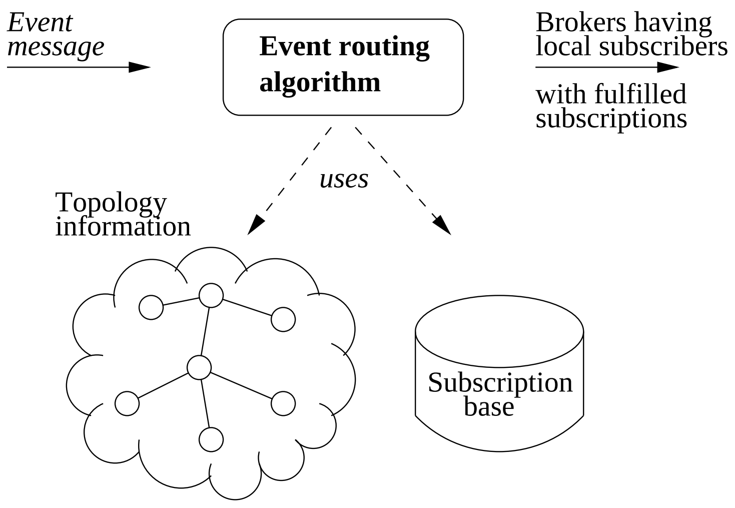


Figure 2.6: Illustration of the task of an event routing algorithm: Based on subscription base and topology information, an incoming event message leads to a set of brokers.

develop new routing algorithms for content-based pub-sub systems but utilize existing approaches. We sketch these approaches and present a justification for our choices in Section 2.4. We illustrate the general task of an event routing algorithm in Figure 2.6.

Let us assume one has been given a particular event routing algorithm. In the literature one can find so-called *event routing optimizations* (or just *routing optimizations*), aimed at improving the un-optimized algorithm with respect to certain parameters or quality measures (e.g., memory usage or system efficiency). Event routing optimizations can be based on subscriptions or on advertisements. Current solutions aim either at reducing the number of routing entries by exploiting redundancies among them, or at subsuming existing routing entries. We describe and evaluate these existing approaches in Section 2.5.

Within this dissertation, we develop novel routing optimizations. Chapter 6 presents our work on subscription-based optimizations; Chapter 7 introduces a novel advertisement-based optimization solution.

Overlapping Task

Content-based pub-sub systems supporting advertisements should exploit the information that is provided by these advertisements. This information, for example, can be used in event routing algorithms. By analyzing the relationships among subscriptions and advertisements, one can decide whether publishers

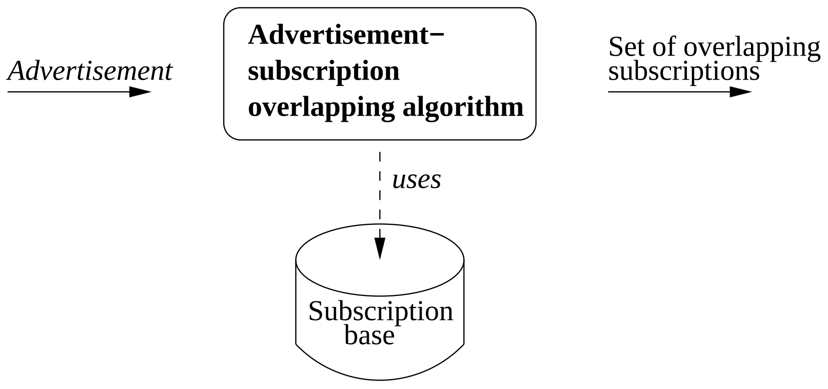

Advertisement
Advertisement−subscription overlapping algorithm
Set of overlapping subscriptions
uses
Subscription base

Figure 2.7: Illustration of the task of an advertisement-subscription overlapping algorithm: Based on the subscription base, an advertisement leads to a set of overlapping subscriptions.

send event messages that overlap the subscriptions of particular subscribers. This relationship is referred to as the *overlapping relationship*. The tasks to solve are the *subscription-advertisement overlapping* and the *advertisement-subscription overlapping task*, depending on the point of view. They are defined as follows:

Definition 2.3 (Advertisement-Subscription Overlapping Task) *The system has been given an advertisement a and a set of subscriptions $\mathcal{S}_i$. It needs to determine all subscriptions $s \in \mathcal{S}_i$ that are fulfilled by any event message conforming to a, that is, $\mathcal{E}(s) \cap \mathcal{E}(a) \neq \emptyset$. We refer to these subscriptions as* overlapping subscriptions*.*

Definition 2.4 (Subscription-Advertisement Overlapping Task) *The system has been given a subscription s and a set of advertisements $\mathcal{A}_i$. It needs to determine all advertisements $a \in \mathcal{A}_i$ having at least one conforming event message that matches s, that is, $\mathcal{E}(s) \cap \mathcal{E}(a) \neq \emptyset$. We refer to these advertisements as* overlapping advertisements*.*

As already mentioned, the solutions to these tasks can be exploited by event routing algorithms, as detailed later. The algorithms to solve these tasks are referred to as *advertisement-subscription overlapping algorithms* and *subscription-advertisement overlapping algorithms*. We illustrate them and their tasks in Figure 2.7 and Figure 2.8. Next to the described formulations of both tasks as function problems, the respective decision problems are required to be solved in pub-sub systems. We go into detail about this distinction for the overlapping task in Chapter 7.

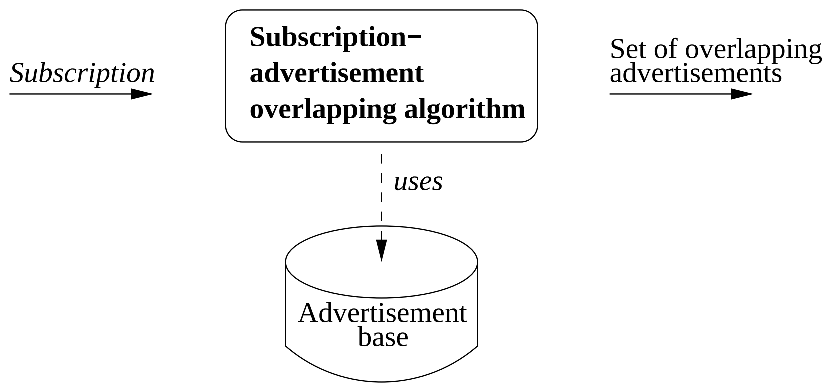

Subscription
Subscription−advertisement overlapping algorithm
Set of overlapping advertisements
uses
Advertisement base

Figure 2.8: Illustration of the task of a subscription-advertisement overlapping algorithm: Based on the advertisement base, a subscription leads to a set of overlapping advertisements.

Extended Tasks

In this dissertation, we focus on those tasks in content-based pub-sub systems we have presented in the three previous subsections. This is due to the insufficiency of existing solutions to these tasks for general application scenarios with respect to general Boolean subscriptions and advertisements, as we reason in detail later. However, we now briefly sketch extended tasks one needs to face in content-based pub-sub systems.

Next to the general event routing task (see page 19) only considering the internal network of a pub-sub system, the system needs to deliver incoming event messages to all subscribers having registered fulfilled subscriptions. This delivery becomes a particularly complex task in cases of large numbers of subscribers, as might be found in ubiquitous computing or sensor networks. Pub-sub systems for such scenarios are addressed, for example, in [HCRW04].

Within this dissertation, we do not consider security aspects in content-based pub-sub systems. They might, however, become highly important when using pub-sub systems commercially. The tasks to be addressed, for example, include not only the filtering of event messages from trusted publishers but also the delivery of these event messages to exclusively trusted subscribers. Furthermore, general confidentiality aspects with respect to event messages and subscriptions need to be considered. Early research on such aspects can be found, for example, in [RR06, WCEW02]. The detection of spam in content-based pub-sub systems, for example, [Tar06], can also be seen as a security issue.

Content-based pub-sub systems should allow for the dynamic reconfiguration of the broker network, including the broker components themselves as well as the links among them. Solutions to these tasks have been presented, for example, in [CMPC03, PCM03]. One can also classify content-based pub-sub systems for general (i.e., constantly changing) peer-to-peer settings in this branch of research, for example, Hermes [Pie04].

We already elaborated on the similarities but also on the differences between content-based pub-sub and database management systems. An idea that emerges immediately when reflecting on this relationship is the support of transactional patterns in content-based pub-sub systems. Preliminary work on this topic can be found, for example, in [MF05].

The final extended research area we want to name here regards the composition of event messages. A composite (or complex) event describes the occurrence of a certain pattern of individual event messages. These messages could, for example, occur one after the other or within a certain time frame. The composition of event messages is addressed, for example, in [PSB04]. An immediate requirement for composite events is the determination of an order of event messages, as described in [LCB99, LSB06].

2.2 Quality Measures, Parameters, and Attributes Influencing Filtering and Routing

Having introduced the general concepts of content-based pub-sub systems and the tasks arising, we now elaborate on the commonly accepted quality measures for pub-sub systems, and the parameters and attributes influencing them.

Quality Measures For Filtering and Routing

We can identify two main quality measures, system efficiency and system scalability, that influence the suitability of content-based pub-sub systems in practice. These general measures largely comply with current assumptions given, for example, in [CCC$^+$01, CRW00, FJL$^+$01]:

By *system efficiency*, we refer to the average time to process an event message by the overall system for a given problem size. We define the processing of an event message e in this context as the task of determining all subscriptions within the distributed system that are fulfilled by e. The problem size in this context refers to the number of registered subscriptions or advertisements.

This measure thus includes both the event filtering and the event routing task, but it excludes the event delivery task. Pub-sub systems aim at high system efficiency, that is, a small processing time per message.

By *system scalability*[1], we refer to the behavior of system efficiency with an increasing problem size. By "problem size" we again refer to the number of subscriptions or advertisements. Thus, this definition focuses on event filtering and routing, but not on event delivery. Our definition of scalability refers to the notions of space-time scalability, as described in [Bon00]. Pub-sub systems aim for sound scalability properties.

Parameters Influencing the Quality Measures

Using these definitions, the two named quality measures have an effect on, and are themselves influenced by, the solutions that are applied to the filtering and routing task in content-based pub-sub systems (see Section 2.1.3):

- Filtering algorithm
- Routing algorithm and routing optimization

These solutions influence two important parameters of pub-sub systems that, in turn, also affect the two quality measures:

- Memory usage
- Network load

We give an overview of the direct dependencies among these quality measures, algorithms, and parameters in Figure 2.9. We describe these dependencies in detail later on.

Internal-Subscription-Model Attribute

An important attribute that, on the one hand, affects the algorithms in pub-sub systems is the internal model of subscription. On the other hand, the internal model of subscriptions influences what algorithms can applied in these systems. We illustrate this twofold effect in Figure 2.9 and elaborate on its occurrence later on (page 27).

[1]By providing our own definition of scalability in this dissertation, we hope to avoid confusion about this term arising from the lack of consensus as to its meaning [DRW06] and the absence of a generally accepted definition [Hil90].

To describe what we mean by the term “internal subscription model”, we have to start by analyzing the use of the notion of expressiveness in the current literature: the term “expressiveness” in the context of pub-sub systems, to our knowledge, has never been properly defined. There are some general explanations, but these fail to provide an acknowledged definition:

Carzaniga and colleagues [CRW00] define expressiveness as the ability of a pub-sub system to express subscriptions[2]. Eugster and colleagues [EFGH02] state that the expressiveness of subscriptions defines how accurately subscriptions can represent the interests of subscribers[3]. Various other work, for example, [AAGC04, BBC$^+$04, CS04, CMPC03, EFGK03, LJ03, PCM03], identify different levels of expressiveness in the distinction between topic-based and content-based pub-sub systems. However, content-based systems (even supporting range queries) can be mapped to topic-based ones, as shown in [TAJ03]. Hence, the term “expressiveness” in the context of pub-sub systems does not model the general notion of expressiveness describing what facts can be represented by a language [MG85].

Li and colleagues [LHJ05] explicitly include the opportunities to combine predicates in subscriptions into their expressiveness definition. They state that in contrast to conjunctive approaches, by providing for arbitrarily complicated Boolean functions in subscriptions, an expressive subscription language can be naturally supported[4]. Mühl [Müh02] also takes this approach and states that the restriction to conjunctions in current pub-sub systems reduces the expressiveness of these systems[5]. These descriptions again show the different use of the term “expressiveness” in the pub-sub context.

To avoid this mismatch between the notion of expressiveness in the general literature and the various notions of expressiveness in the pub-sub area, we refer to the concept of the “expressiveness of a subscription language” (in terms of pub-sub) as *internal subscription model* in the following (we similarly use

[2] “Expressiveness refers to the power of the data model that is offered to publishers and subscribers of notifications.” [CRW00]

[3] “The expressiveness of subscriptions defines how accurately subscriptions can represent the interests of the subscribers. With different kinds of subscription languages, it is possible to achieve different ‘levels’ of expressiveness.” [EFGH02]

[4] “Siena and Jedi exploit covering-based routing. Unfortunately, they restrict the expressiveness of content-based routing, and do not consider merging techniques. . . . Since BDDs can be used to represent arbitrarily complicated Boolean functions, an expressive subscription language can be naturally supported.” [LHJ05]

[5] “Siena and Rebeca restrict filters to be conjunctions of attribute filters. On one hand, this restriction reduces the expressiveness of the filter model, but on the other hand it enables routing optimizations like covering to be applied efficiently.” [Müh02]

the term *internal advertisement model* for advertisements). For content-based pub-sub systems, we can distinguish between subscriptions and advertisements as purely conjunctive filter expressions, and subscriptions and advertisements as general Boolean filter expressions.

Applicability Attribute

As we already outlined previously, the focus in this dissertation is on general-purpose pub-sub systems, as opposed to system solutions for a particular application setting. The filtering and routing solutions applied in such a system, therefore, need to constitute generic approaches to filtering and routing. Clearly, the suitability as a general-purpose solution is not contradicted if a filtering or routing approach effectively exploits certain application-specific attributes. As long as such attributes are only exploited by an algorithm, but their absence does not impair the functioning of the algorithm, that is, these attributes are no mandatory requirement, this algorithm classifies as a general-purpose solution.

However, if, for example, a filtering algorithm is entirely restricted to a certain specific application, we do not consider this algorithm a general-purpose solution. A suitability as a general-purpose approach is also not given if, for example, in general settings[6] the space or time efficiency properties of a filtering algorithm degrade to those of a basic approach and contradict its original design goals. Furthermore, we do not consider solutions to be generally applicable if they merely represent a static system solution, for example, filtering algorithms that, due to their inherent structure, cannot efficiently register or deregister subscriptions.

We also refer to the attribute of the suitability of a solution as a general-purpose approach by the term *applicability*. In accordance with the current practice, we consider subscriptions as highly selective filter expressions. That is, usually only a small proportion of messages fulfills a registered subscription. Thus, the consideration of such an application scenario does not oppose the applicability attribute.

We included this applicability attribute into our overview of the dependencies among quality measures, algorithms, and parameters in Figure 2.9. The illustrated cross-influences are caused by the following observations.

[6]These general settings should obviously have reasonable assumptions.

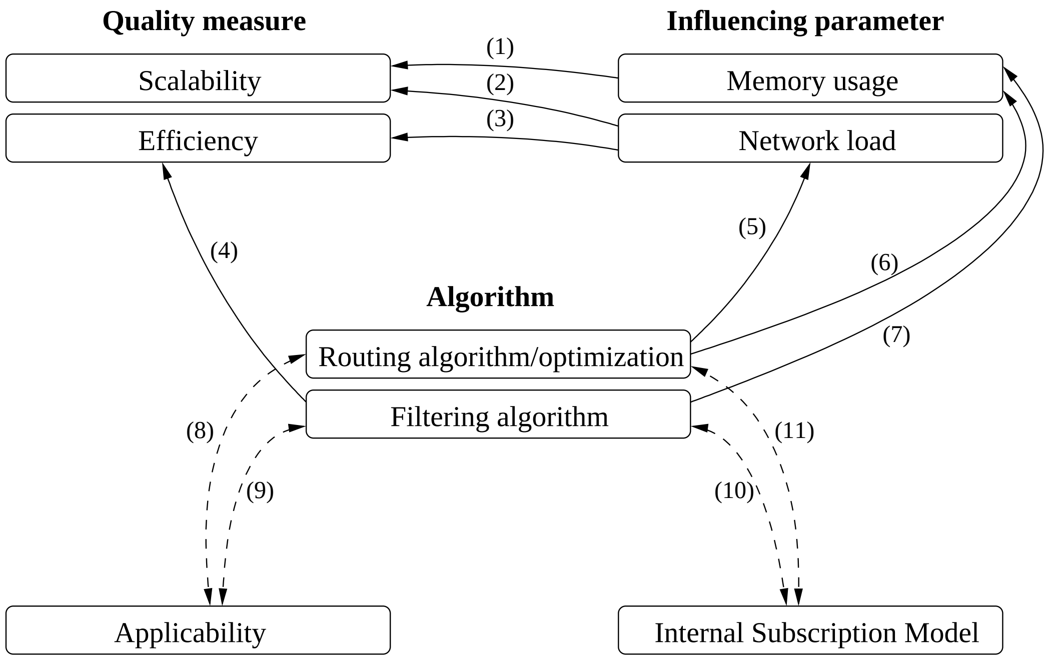


Figure 2.9: Overview of the cross-influences among quality measures, algorithms, and parameters. We named these influences to be able to reference them.

Dependencies Among Quality Measures, Algorithms, and Parameters

All recent content-based pub-sub systems apply main memory filtering algorithms to achieve a high system efficiency. This development has become feasible due to the employment of cheap, large main memories in computers. The result is an efficient event filtering in individual broker components. Evidently, the applied main memory filtering algorithm still plays an extremely relevant part regarding filter efficiency (Influence 4 in Figure 2.9). The efficiency of the overall distributed system, however, depends on the applied event routing algorithm and optimization as well, due to their influence on the network load (Influence 5 in conjunction with Influence 3 in Figure 2.9).

Although large main memories are a standard today, filtering algorithms should require as few memory resources as possible. It is the one influence on space scalability [Bon00] and a crucial influence on space-time scalability in individual broker components (Influence 1 in Figure 2.9). The less memory the filtering algorithm demands per subscription and advertisement, the more subscriptions and advertisements are supported. Thus a filtering algorithm requiring less memory than another one achieving the same efficiency is the preferred choice. Requiring too much memory resource, on the other hand, leads to frequent page swaps, degrading the achieved system efficiency by several orders of magnitude.

The overall scalability of a content-based pub-sub system additionally depends on the utilized event routing algorithm, partially determining the sizes of routing tables, the complexity of routing table entries (Influence 6 in Figure 2.9 for both of them), and the number of routed messages (Influence 5 in Figure 2.9). These properties influence each other. For example, an increase in internally routed (and processed) event messages with a simultaneous decrease in the complexity of routing entries might improve the overall system efficiency. The number of routed event messages, however, is the strongest influence on system scalability if assuming limited network resources (Influence 2 in Figure 2.9). Next to the applied routing algorithm, the utilized routing optimization strongly affects overall scalability (Influences 6 and 5 in conjunction with Influences 1 and 2, respectively, in Figure 2.9).

The internal subscription model affects the choice of a filtering algorithm and thus implies memory usage and filter efficiency. That is, the internal subscription model has an indirect effect on both scalability (Influences 10, 7, and 1 in Figure 2.9) and efficiency (Influences 10 and 4 in Figure 2.9), the two quality measures. For the other direction, the internal subscription model of a system, obviously, has to be supported by the applied routing algorithm and optimization (Influence 11 in Figure 2.9), and the filtering algorithm used (Influence 10 in Figure 2.9).

With respect to applicability, filtering algorithm, routing algorithm, and routing optimization have to fulfill this attribute (Influences 8 and 9 in Figure 2.9). These algorithms directly influence the quality measures of the system, as illustrated in Figure 2.9 and stated in our definition of applicability. Considering the other direction, the applied algorithms either constitute general-purpose approaches or specialized solutions, that is, based on their internal functioning a general applicability is given or not given.

In the following section, we present the state of the art for solutions to one of the introduced tasks, the filtering algorithm. As can be seen in the figure, the applied filtering algorithm influences the two identified quality measures (system efficiency and scalability). Furthermore, the filtering algorithm needs to fulfill the general-purpose attribute and supports a particular internal subscription model.

In Section 2.4, we then present current event routing algorithms, followed by an analysis of existing routing optimizations in Section 2.5 (solutions to the other task). Routing algorithm and optimization indirectly influence both quality measures, as illustrated in Figure 2.9. Later on, we develop novel solutions (for the filtering and routing task) and show their effects on the identified parameters and quality measures.

2.3 Event Filtering Algorithms

Having introduced the general foundations of pub-sub systems and the widely applied quality measures, we now elaborate on the solution to one task that affects these measures: the utilized event filtering algorithm. In this section, we subsequently present a categorization of existing filtering solutions based on the applied predicate and subscription indexing approach (Section 2.3.1).

We then analyze the main representatives of filtering algorithms with respect to the identified quality measures and the applicability attribute in Section 2.3.2. As we will discover in the analysis, only the conjunctive counting algorithm classifies as a general-purpose filtering approach that targets both quality measures as its design goals. We sketch this algorithm in detail in Section 2.3.3.

2.3.1 Categorization of Filtering Algorithms

Pub-sub systems usually apply a kind of subscription indexing and predicate indexing to allow for an efficient event filtering process. We can use these two kinds of indexing as two dimensions for a classification of current filtering algorithms. With respect to predicate indexing, we can differentiate between:

- *no predicate indexing* approaches (NP), and
- *one-dimensional predicate indexing* approaches (OP).

With respect to subscription indexing, we can identify:

- *individual subscription indexing* approaches (IS), and
- *shared subscription indexing* approaches (SS).

Description of Algorithm Categories

No predicate indexing approaches (NP) do not apply designated data structures to efficiently determine whether predicates of subscriptions are fulfilled by the attribute-value pairs of an incoming message. One-dimensional predicate indexing approaches (OP), on the other hand, apply predicate indexes on a per-attribute basis. That is, all predicates that specify a certain attribute are included in this index. Predicate indexes might additionally be specialized according to the filter function that is used by predicates, leading to several predicate indexes per attribute.

Individual subscription indexing approaches (IS) store subscriptions in a way that allows for the efficient determination of fulfilled subscriptions, for example, based on information about their predicates. Each subscription is indexed individually without considering already indexed subscriptions. Shared subscription indexing approaches (SS), on the other hand, aim at compacting several subscriptions. Variations range from the usage of exactly one subscription index structure to represent all registered subscriptions, to the use of various subscription index structures to represent all registered subscriptions.

By combining these two dimensions for classifying filtering algorithms, we derive four categories of algorithms: NP-IS, NP-SS, OP-IS, and OP-SS. We give the main representatives of these categories in the following paragraphs.

Representatives of Algorithm Categories

An example of NP-IS is Elvin [SA97, SAB$^+$00]: there are no predicate indexes and individual subscriptions are evaluated against incoming messages. This idea constitutes a basic solution to the filtering task. However, Elvin supports a general Boolean subscription language and the filter functions in predicates are more sophisticated than in other systems.

The approaches from Aguilera and colleagues [ASS$^+$99], Gough and Smith [GS95], and Campailla and colleagues [CCC$^+$01] classify as NP-SS. The former two approaches use tree structures to represent the registered conjunctive subscriptions. The latter approach uses a shared binary decision diagram with several output nodes, a graph structure, as its subscription index.

The counting algorithm [AJL02, YGM94] as well as the cluster algorithm [HCH$^+$99, FJL$^+$01] fall into category OP-IS. These approaches index both predicates and subscriptions and are restricted to subscriptions in conjunctive form. The counting algorithm gets its name from counting the number of fulfilled predicates per subscription when filtering. The cluster algorithm gets its name from clustering subscriptions according to their number of predicates and their use of common equality predicates.

Finally, the approach in [LHJ05] belongs to category OP-SS. It indexes predicates according to the applied operators and attributes, and uses modified binary decision diagrams as subscription index structures for restricted conjunctive subscriptions.

In the following subsection, we analyze the suitability of these algorithms for general-purpose application scenarios.

2.3.2 Applicability of Algorithms

All of the previously mentioned algorithms provide sound solutions to the filtering task, provided their assumptions about application specifics are met. Their individual design goals in their target application setting are to realize a particular space-efficient filtering process, to realize a particular time-efficient filtering process, or to offer a flexible subscription language to their users. The ranges of settings that these algorithms have been developed for vary in their width. Generally the algorithms gain their benefits with respect to filter efficiency or memory usage by exploiting the specifics of those scenarios they have been designed for.

Considering our requirement of a general-purpose algorithm, most approaches become unsuitable with respect to either their memory requirements or their filter efficiency if the application specifics that are exploited in the filtering process do not hold. In the following paragraphs, we analyze these algorithms according to the four identified algorithm categories.

No Predicate Indexing, Individual Subscription Indexing Approaches

As we introduced previously, the Elvin system [SA97, SAB$^+$00] falls into category NP-IS. The following analysis shows that this filtering approach does not constitute a general-purpose solution.

Elvin [SA97, SAB$^+$00]. The filtering approach of Elvin is usually described as a “naïve” (e.g., [LHJ05, PFLS00]) or “brute-force” (e.g., [MFP06, TKD04, YGM99]) solution to the filtering task in the literature. This description stems from the fact that all subscriptions, including their predicates, are individually considered by this approach. Evidently, this method does not lead to high filter efficiency. In particular, the approach does not scale to a growing number of registered subscriptions, and thus does not constitute a generic filtering solution. This limitation of [SA97, SAB+00] with respect to scaling to a large subscription base is also identified in the literature, for example, [CRW01]. However, Elvin internally supports a more general subscription language (with respect to both the supported predicates and their combination) than other systems.

Generally the approach of individually analyzing all registered subscriptions is relatively well-suited for scenarios where most subscriptions match most incoming messages. Here nearly all subscriptions need to be fully analyzed by any current filtering approach in order to determine whether they are fulfilled by the incoming message. There is no criterion that would allow a recent algorithm to stop the evaluation of a subscription after its partial analysis. Evidently this narrow application scenario does not fulfill the general-purpose requirement and contradicts typical assumptions about the selectivity of subscriptions (see Section 2.2).

No Predicate Indexing, Shared Subscription Indexing Approaches

We identified three main algorithm representatives in category NP-SS. All of them do not classify as general-purpose filtering solutions, because the supported application fields are too narrow and a widening results in a degeneration to a basic filtering approach.

Gough and Smith [GS95]. This tree-based conjunctive filtering algorithm generally leads to high filter efficiency due to the approach of traversing exactly one path in the created subscription index tree for an incoming message. However, this advantage is firstly counteracted by the limited range of operators that is supported by this approach: [GS95] effectively supports equality predicates only. Range tests and set membership tests can be supported by the approach. However, this extension comes at the cost of strongly growing memory requirements. These high memory requirements are also noted as a major restriction of the algorithm in the literature, for example, [ASS+99, FJL+01, RDJ02, WK05].

Additionally, the created subscription index tree cannot effectively handle subscriptions that do not specify all attributes in their predicates. Although such subscriptions can be extended by *"don't-care"* predicates (fulfilled by all possible attribute values) and inserted into the index structure, each "don't-care branch" in a node of the index tree needs to contain a combination of all subtrees that can be reached by the other branches (no "don't-care" predicates) of that node. This behavior, evidently, leads to a potentially exponential explosion in the size of the index tree [ASS$^+$99].

Another restricting attribute is that the index tree requires costly preprocessing to handle registrations and deregistrations of subscriptions. [GS95] thus presents a static solution to the filtering task, which has been identified as another shortcoming of the approach in the literature, for example, [FJL$^+$01, MFP06, RDJ02, WK05]. On this basis, we conclude that the algorithm does not classify as a generic solution to the filtering task.

Aguilera and colleagues [ASS$^+$99]. Aguilera and colleagues [ASS$^+$99] present another tree-based filtering approach for conjunctive subscriptions. It aims at solving some of the problems of [GS95] with respect to memory requirements. However, this attempt comes at the cost of time efficiency because it cannot filter messages by following one path in the subscription index tree anymore. Nevertheless, the approach [ASS$^+$99] is still characterized as too memory consuming, for example, [FJL$^+$01, RDJ02, WK05].

Aguilera and colleagues focus on supporting equality predicates [MFP06] in subscriptions; they present some optimizations for this restricted setting to reduce the size of the tree. The overall approach in [ASS$^+$99] might generally be applicable to operators other than equality as well. However, in this case, the subscription index tree increases in both height and width, sharing the problems of [GS95].

Predicates are not indexed in this approach. Subscriptions can only be shared in a branch of the index tree if all predicates of these subscriptions are the same from a particular point onwards. Furthermore, the order of attributes and operators needs to be predefined, which is too strong an assumption for a general-purpose solution. Because various branches of the created index tree need to be evaluated in the filtering process and predicates are not indexed, the approach in general degrades in efficiency for general settings.

Moreover, the insertion process for newly registered subscriptions is highly inefficient if the optimizations that are required to avoid an explosion in tree size are applied: nearly the whole tree might have to be analyzed for insertion, making it a static filtering solution in practice. This limitation is identified in the literature, for example, [FJL$^+$01, RDJ02, WK05].

We therefore conclude that [ASS$^+$99] is not a suitable approach for general applications.

Campailla and colleagues [CCC$^+$01]. Next to Elvin [SA97, SAB$^+$00], this approach is the only filtering algorithm that supports general Boolean subscriptions. Its main idea is to represent subscriptions by an ordered binary decision diagram (BDD) [Bry86].

In the filtering algorithm, under all circumstances, all registered subscriptions need to be fully evaluated for each incoming message. This is because the subscription index is evaluated backwards, from the terminal nodes (no children) to the output nodes (no parents) of the BDD. This attribute only makes it a feasible solution if the created BDDs for subscriptions represent graphs with highly equivalent subgraphs in their lower parts. Otherwise, the algorithm degrades to the basic filtering approach, not fulfilling our (and its own) requirements. Additionally, predicates are not indexed, leading to a costly predicate evaluation process in general.

It is hence not only an assumption that subscriptions are highly similar with respect to both their predicates and the combination of these predicates. Moreover, it is a requirement that these redundancies can be exploited in the created subscription index. As we will demonstrate later, the experiments in [CCC$^+$01] show that the presented approach does not fulfill this goal even if its strong redundancy assumption is met.

With respect to memory requirements, the size of a BDD, and thus the size of the subscription index, may become exponential [CCC$^+$01, MFP06] (general Boolean subscriptions are supported). The size in practice strongly depends on the ordering of variables (predicates of subscription). The determination of an optimal order is an NP-hard problem [BW96]. [CCC$^+$01] does not consider the ordering of variables. Instead, the approach requires a given, fixed order of variables. If this order needs to change (e.g., because new subscriptions have been registered), all subscriptions have to be re-indexed by the approach, making it impractical to adapt to the current subscription set. The approach outlined in [CCC$^+$01] thus also constitutes a restricted filtering solution in this respect. Furthermore, [CCC$^+$01] leaves open the question of how a newly registered subscription is inserted into a BDD, this being one of the key points for the construction and the size of the subscription index.

Nevertheless, let us assume the restricted application setting that suits this approach. The experiments in [CCC$^+$01] assume a total of only 208 distinct predicates within all registered subscriptions. The experimental evaluation reveals that even such highly similar subscriptions already lead to linearly growing sizes of the subscription index with an increasing subscription number, and thus to linearly increasing filtering times. If these specialized experiments show a linear increase in the size of the index structure, more general application settings are expected to lead to index sizes, and thus filtering times, that grow exponentially with the number of registered subscriptions (general Boolean subscriptions are supported).

Moreover, in the dynamic version of the system that does not require an iterative (and thus costly) index minimization on a per-subscription basis, the number of nodes in the created subscription index is only marginally less than the average number of predicates per subscription (7.09 nodes compared to 7.6 predicates per subscription on average). Thus, even in experiments with highly redundant subscriptions, [CCC$^+$01] cannot exploit existing redundancies among subscriptions.

We conclude that [CCC$^+$01] leaves open too many fundamental questions, does not address our general-purpose requirement, and cannot even exploit the redundancy among highly common subscriptions.

One-Dimensional Predicate Indexing, Shared Subscription Indexing Approaches

Only one main algorithm falls into category OP-SS, [LHJ05]. As we demonstrate, this approach is also too restricted in its applicability to be considered a general-purpose filtering algorithm.

Li and colleagues [LHJ05]. This filtering algorithm, sketched in [LHJ05], was proposed concurrently to our work (Chapter 4); it uses modified binary decision diagrams (MBDs) [JT92] as subscription index structure. In contrast to the original BDD approach presented in [CCC$^+$01], [LHJ05] is restricted to conjunctive subscriptions. Another difference is that the registered subscriptions are represented by a set of MBDs, that is, several indexes represent registered subscriptions. As an extension of [CCC$^+$01], [LHJ05] applies one-dimensional predicate index structures, resulting in its classification as OP.

Despite these differences, [LHJ05] shares the problems and limitations of [CCC$^+$01] that have been described in the previous subsection: For each incoming event message, all registered subscriptions need to be fully analyzed.

Even though it can be decided whether predicates are fulfilled by incoming messages by consulting the predicate index structures, all MBDs still have to be completely evaluated to determine fulfilled subscriptions. This requirement of the full evaluation of all MBDs for each message is a substantial drawback of [LHJ05] (as well as [CCC$^+$01]): the complexity of filtering any message in [LHJ05] directly corresponds to the size of the subscription index structure. As the original BDD solution, [LHJ05] thus degrades to the basic filtering approach, if the presumed high redundancy among subscriptions is not given or the created subscription index cannot exploit the existing redundancies.

One of the open points of the original BDD approach [CCC$^+$01] is how newly registered subscriptions are inserted into the existing subscription index structure. [LHJ05] tries to exploit its restriction to conjunctive forms to decide whether a newly registered subscription is integrated into an existing MBD or is inserted as a new MBD in the subscription index. The presented insertion approach, however, depends on a given fixed order of variables. Thus, the algorithm still (as the original approach) requires the impractical re-indexing of all subscriptions if the globally chosen order becomes suboptimal. Therefore, the suitability of [LHJ05] for non-static environments with potentially changing characteristics of subscriptions is not given.

Generally the method of ordering variables that is proposed in [LHJ05] is not applicable to general settings with non-extreme predicate redundancy. It is even inapplicable if only a marginal proportion of predicates is not shared among most subscriptions. The reason for this property is that MBDs in [LHJ05] can only be shared by those subscriptions that specify the same second predicate according to the given attribute order. However, the ordering required by [LHJ05] starts with the least common predicates because redundancies among subscriptions can only be exploited at the bottom of MBDs. The sharing of MBDs thus breaks down as soon as subscriptions do not contain highly common predicates only. It is noteworthy that even in the best possible case only those parts of subscriptions that contain exactly the same predicates from a particular point onwards (in the assumed, fixed predicate order) can be shared.

Thus, as with the original BDD approach, the solution in [LHJ05] does not constitute a general-purpose filtering algorithm. Additionally only excerpts of the algorithm are briefly sketched in [LHJ05]—the work mainly focuses on event routing optimizations. [LHJ05] does not even investigate the size of the created MBDs. There is no reason why the arbitrary selection of the second predicate (in a fixed order) for sharing MBDs should lead to better results than [CCC$^+$01]. The settings analyzed in [LHJ05] are restricted and contain highly redundant predicates: One data set contains 2,000 distinct predicates; another data set contains 5,000 distinct predicates.

One-Dimensional Predicate Indexing, Individual Subscription Indexing Approaches

For category OP-IS, we named two filtering algorithms in Section 2.3.1. Only one of them, the counting approach, constitutes a general-purpose solution to the filtering task, as described in the following paragraphs.

Cluster algorithm [HCH$^+$99, FJL$^+$01]. This conjunctive filtering algorithm applies one-dimensional predicate indexes for efficiency reasons. The approach is presented in detail by Fabret and colleagues [FJL$^+$01] and is based on a proposal by Hanson and colleagues [HCH$^+$99]. Its general idea is to cluster sets of subscriptions. However, as we describe later on, the criterion required for an effective clustering disqualifies [FJL$^+$01] as a general-purpose solution.

With respect to efficient filtering, [FJL$^+$01] proposes to cluster subscriptions in such a way that for each incoming event message only a minimal number of clusters (preferably one cluster) can contain fulfilled subscriptions. Hence, the clusters that are determined for an incoming message usually include both fulfilled and unfulfilled subscriptions. In order to derive the set of fulfilled subscriptions from each cluster, all subscriptions in this cluster are evaluated by the filtering algorithm. In combination with the applied predicate indexes, it is sufficient to analyze whether all predicates of each subscription are fulfilled.

[FJL$^+$01] uses the notion of *access predicate* to refer to a predicate (or a set of predicates) that is used for clustering. The approach considers equality predicates as access predicates [MFP06][7]. This assumption on its own already disqualifies [FJL$^+$01] as a general-purpose solution because it is not applicable in other scenarios at all. Furthermore, “intricate schemes” are required by the cluster algorithm to determine access predicates, as admitted by one of its authors [AJL02].

[7][FJL$^+$01] states that a further property of access predicates is that they are required to be fulfilled in a fulfilled subscription. Evidently, this is the case for all predicates because [FJL$^+$01] is restricted to conjunctive subscriptions.

But even if assuming that there is at least one equality predicate per subscription, [FJL+01] only achieves an appropriate filter efficiency if the majority of predicates in subscriptions are equality predicates. Only if this strong assumption holds, does the clustering envisaged by [FJL+01] become possible. Furthermore, the algorithm requires subscriptions to contain the same overall number of predicates to be able to cluster them together.

These problems let us conclude that [FJL+01] only constitutes an appropriate filtering solution in highly limited application settings.

Counting algorithm [AJL02, YGM94]. The counting algorithm is a filtering approach for conjunctive subscriptions that balances memory usage and filter efficiency, and fulfills the requirement of its applicability in a wide range of settings. We give a technical description of the counting approach in Section 2.3.3. In the following paragraphs, we demonstrate its broad idea and the resulting suitability for various scenarios.

The overall idea of the counting algorithm is to count the number of fulfilled predicates per subscription in the filtering process. The counting of predicates is based on one-dimensional predicate indexes, allowing for the determination of all predicates that are fulfilled by an incoming message. Having counted the number of fulfilled predicates per subscription, all those subscriptions whose counter equals their overall number of predicates constitute a fulfilled subscription.

Analyzing this approach to filtering, we firstly realize that the counting algorithm is independent of the redundancy among predicates. Secondly, it is applicable regardless of the similarity among subscriptions. Thirdly, [AJL02, YGM94] does not depend on the use of particular attribute filters in predicates (considering their effect on subscription indexing). Conversely, the approach shows comparable filter efficiency and memory requirements for a wide range of settings independently of the previous mentioned parameters. Fourthly, the created subscription index is highly flexible with respect to both registrations and deregistrations, and changing subscription characteristics. Altogether these characteristics make the counting approach a general-purpose filtering solution.

Evidently, [AJL02, YGM94] does not represent the most time-efficient and the most space-efficient filtering solution in those specialized settings that are (either exclusively or primarily) targeted by the previously analyzed filtering solutions. However, as identified, the counting approach represents a general-purpose solution that, firstly, can be applied to the full range of settings. Secondly, its time efficiency properties do not degrade to a basic filtering approach if the exploited parameter setting does not hold. Thirdly, its memory requirements remain stable over all settings and do not grow excessively for general scenarios.

Summary

Our analysis in this section led to two main findings:

1. There exist only basic filtering algorithms for general Boolean subscriptions.

2. All conjunctive filtering algorithms except one have too strong requirements on the supported application scenario to classify as a general-purpose solution.

With respect to Finding 1, there are two existing approaches for the filtering of general Boolean subscriptions. Both do not apply predicate index structures, requiring the individual consideration of each predicate. Elvin [SA97, SAB+00] constitutes the basic filtering algorithm, additionally requiring the consideration of each individual subscription in the filtering process. Campailla and colleagues [CCC+01], on the other hand, cannot adapt to changes in the subscription base. Although the approach indexes subscriptions, all subscriptions need to be fully evaluated in the filtering process. Already for scenarios with highly similar subscriptions, [CCC+01] cannot exploit existing redundancies, de facto leading to the same problem size as in the basic filtering approach.

With respect to Finding 2, most existing conjunctive filtering algorithms are designed to exploit particular patterns of “niche” application scenarios. Some of these algorithms cannot be applied to more general settings at all. Other algorithms are generally applicable to a broader range of scenarios. However, either their internal filtering process becomes the basic approach in this case, or the memory requirements of the algorithm explode exponentially.

As we outlined before, in this dissertation we focus on solutions that are applicable to a wide range of application scenarios. In our previous analysis, it became evident that only the counting algorithm [AJL02, YGM94], an individual subscription indexing approach, constitutes a filtering algorithm that is well-suited for general application settings. Existing shared subscription indexing approaches, on the other hand, are too restrictive in their assumptions and not applicable to general application scenarios.

Taking into account these findings, we implicitly refer to individual subscription indexing approaches in general and to the counting algorithm in particular when talking about filtering algorithms in the following.

Based on our requirement of a general-purpose filtering solution for general Boolean subscriptions, we ultimately extend the conjunctive counting algorithm to a general Boolean solution in Chapter 4. In the following section, we give a technical outline of the original counting approach to give the reader a better understanding of this algorithm.

2.3.3 Outline of the Counting Algorithm

As previously identified, the counting algorithm [AJL02, YGM94] is a general-purpose conjunctive algorithm. It was originally proposed by Yan and Garcia-Molina in [YGM94]. According to our categorization, the counting algorithm classifies as a one-dimensional predicate indexing approach. Internally, the algorithm assigns artificial identifiers to all predicates p and conjunctive subscriptions s: id(p) and id(s), respectively.

The event filtering process in the counting algorithm comprises two steps as follows: In the first filtering step, *predicate matching*, the algorithm determines all fulfilled predicates for the incoming event message e. These predicates are obtained by using the one-dimensional predicate indexes for every attribute-value pair of e; this information is recorded in a *fulfilled predicate vector*.

The second filtering step, *subscription matching*, calculates all fulfilled subscriptions based on the information obtained previously and stored in the fulfilled predicate vector. We illustrate the subscription matching step in Figure 2.10. Its description in written form is as follows:

In the first phase, the algorithm accumulates a counter in a *hit vector*, containing one entry per registered subscription. Information about the use of predicates in subscriptions is found in a *predicate-subscription association table*, built up when registering subscriptions. This table associates every predicate identifier id(p) that is known to the system to a set of subscription identifiers $\{id(s_i), \ldots, id(s_j)\}$. The semantics is that the respective predicate is contained in every subscription it is associated with in this table.

After all fulfilled predicates have been calculated and their counters have

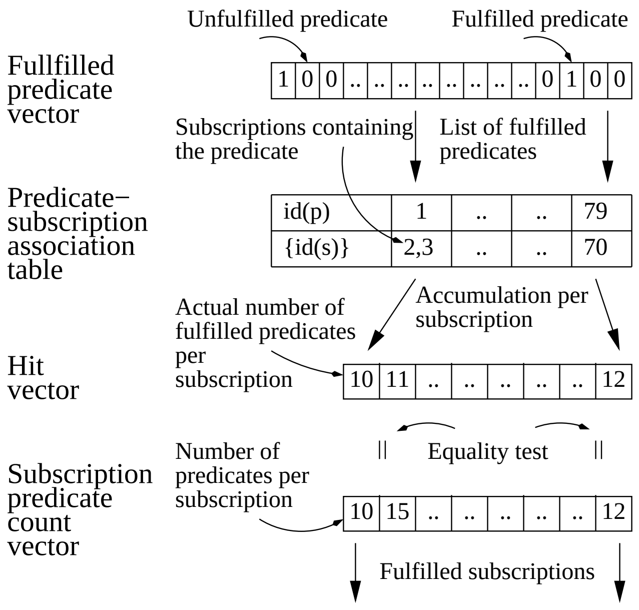

Unfulfilled predicate
Fulfilled predicate
Fullfilled predicate vector
1 0 0 .. .. 0 1 0 0
Subscriptions containing the predicate
List of fulfilled predicates
Predicate–subscription association table
id(p) 1 79
{id(s)} 2,3 70
Accumulation per subscription
Actual number of fulfilled predicates per subscription
Hit vector
10 11 12
Equality test
Number of predicates per subscription
Subscription predicate count vector
10 15 12
Fulfilled subscriptions

Figure 2.10: Overview of the subscription matching step and the required matching structures in the conjunctive counting algorithm. We use integers as example predicate and subscription identifiers.

been increased, the hit vector contains the number of fulfilled predicates per conjunctive subscription. To determine the fulfilled subscriptions, the algorithm now exploits the restriction of subscriptions to conjunctive forms: only if exactly all predicates of a subscription s are fulfilled, s is fulfilled as well. In the last phase, the algorithm compares the counter accumulated in the *hit vector* to the overall number of predicates for each subscription. This information about the number of predicates is stored in a *subscription predicate count vector*, which is populated when subscriptions are registered. If both counters show the same value, the respective conjunctive subscription is fulfilled. A graphic overview of this subscription matching step is pictured in Figure 2.10. Here we illustrate the fulfilled predicate vector using a bit vector implementation.

The counting algorithm additionally requires a means to determine all predicates that are contained in a conjunctive subscription. This information is needed to efficiently support deregistrations. One approach to provide this information is the use of a *subscription-predicate association table*, as proposed by Ashayer and colleagues in [AJL02]. The basic solution to search through all entries in the predicate-subscription association table is not suitable in practice due to efficiency aspects.

Part 1 of our central hypothesis (page 6) regards the unsuitability of general-purpose conjunctive filtering algorithms for general Boolean subscriptions. We give details about the effects of the required conversion of general Boolean subscriptions to conjunctive subscriptions in Section 2.6. Before proceeding to these descriptions, we provide information about routing algorithms and routing optimizations for pub-sub systems in the following two sections.

2.4 Event Routing Algorithms

Having elaborated on current solutions to the event filtering task in the last section, we now present the state-of-the-art for the second task in content-based pub-sub systems, the event routing task (see Section 2.1.3). Within this dissertation, we do not work on event routing algorithms. Instead, we apply existing solutions but propose novel event routing optimizations (see Section 2.5 for an overview of existing optimizations), subscription pruning (Chapter 6) and advertisement pruning (Chapter 7).

In the following descriptions, we build on our assumptions about the structure of distributed content-based pub-sub systems, as we stated in Section 2.1.2 (page 17). The algorithms described in the following introduce the general ideas of existing routing approaches. We neither present all details of particular implementations nor analyze the differences between the systems applying them. Within this section, we just present those particulars that are required to understand the contributions of this dissertation. A detailed classification and analysis of event routing algorithms can be found elsewhere, for example, in [BH04, MFP06].

2.4.1 Event Forwarding

Considering our definition of the event routing task (see Definition 2.2 on page 19), the *event forwarding* algorithm [MFP06] (referred to as flooding in that work) solves this task, but it does so in a very network-consuming way. It merely distributes all event messages within the whole network of brokers. That is, it also routes a message e to those brokers that have not registered local subscriptions fulfilled by e. Hence, the event forwarding approach does not even require the construction of routing tables and, therefore, is one of the simplest routing algorithms for content-based pub-sub systems.

Following the common assumption of an acyclic overlay network as connection among brokers (see Section 2.1.2), the event forwarding routing algorithm works as follows. We here consider the general case and do not distinguish between event messages published by local publishers and messages routed by brokers. We refer to the originator of a message as *sender*:

- Forward an incoming event message e to all neighbor brokers in the network except the sender of e.

As one can realize, the event forwarding approach is straightforward conceptually and also in its implementation, and it solves the event routing task. However, the drawback of event forwarding is the created network load: all messages are basically flooded within the whole network of brokers [BH04].

2.4.2 Subscription Forwarding

Using *subscription forwarding* as the routing algorithm offers a means to avoid the flooding of event messages within the network of brokers. This approach, however, involves a specialized handling of subscriptions: whenever a subscription s is registered by subscriber S, the designated local broker $B(S)$ distributes s to its neighbor brokers. These brokers, in turn, send s to all of their neighbors except the sending broker. Proceeding in that way, subscriptions construct what we refer to as *event routing tables*, required in the event routing process.

This routing of event messages when applying subscription forwarding is based purely on the created event routing tables, and it works as follows (we again consider the general case as in Section 2.4.1):

- Forward an incoming event message e to all neighbor brokers that previously sent subscriptions fulfilled by e.

The subscription forwarding approach also requires a specialized handling of deregistrations of subscriptions: the deregistration information needs to be distributed to other brokers. We neither elaborate on this aspect nor provide further details here, because routing algorithms are out of the main focus of this dissertation and not required for the understanding of our contributions. Instead, we refer interested readers to the original work on subscription forwarding, for example, by Carzaniga and colleagues [CRW01] and by Mühl [Müh02].

2.4.3 Advertisement Forwarding

Advertisement forwarding also prevents the flooding of event messages within a content-based pub-sub system. Additionally, it avoids the forwarding of all subscriptions in the network. Instead, advertisement forwarding distributes advertisements and only selectively forwards subscriptions based on the distributed advertisement information.

An advertisement a is forwarded to all neighbors by the designated local broker and to all neighbors except the sender by non-local brokers. This procedure is analogous to the distribution of subscriptions in the subscription forwarding algorithm (see Section 2.4.2). These forwarded advertisements represent entries in *subscription routing tables*.

The distribution of subscriptions in the advertisement forwarding approach is purely based on the created subscription routing tables: The algorithm only distributes subscriptions to those neighbor brokers that previously forwarded at least one overlapping advertisement. These subscriptions again create *event routing tables*, used in the event routing process. Compared to subscription forwarding, the advertisement forwarding algorithm therefore decreases the sizes of the event routing tables, but it additionally requires the creation of subscription routing tables. Considering both algorithms, one could conclude that advertisement forwarding is an extension of the subscription forwarding approach if incorporating advertisements.

The routing of event messages is again based purely on the created event routing tables and works as follows (again, we consider the general case as in Section 2.4.1):

- Forward every incoming event message e to all neighbor brokers that previously sent subscriptions fulfilled by e.

The advertisement forwarding approach requires a specialized handling of deregistrations of advertisements and subscriptions, for example, the distribution of the deregistration of a subscription and the distribution of the deregistration of an advertisement (which additionally leads to the removal of entries in event routing tables). Furthermore, the registration of advertisements, under these circumstances, requires a distribution of its overlapping subscriptions. We do not elaborate on these details here. Interested readers are referred to the original work on advertisement forwarding, for example, described by Carzaniga and colleagues [CRW01] or, in more detail, by Mühl [Müh02].

2.4.4 Rendezvous Brokers

The routing by the application of *rendezvous brokers* also avoids the flooding of event messages within the broker network. Rendezvous brokers are a meeting point for subscriptions and advertisements [PB02] of a particular *type*[8]. Every rendezvous broker is responsible for one or several event types. There is one active rendezvous broker for each supported type[9].

To allow brokers to act as rendezvous points, all subscriptions and advertisements of a particular type are sent towards the designated rendezvous broker. In this way, forwarding broker components and the rendezvous broker itself integrate information about advertisements into their *subscription routing tables*[10]. The same holds for subscriptions: Brokers on the path to the rendezvous point, and the rendezvous broker, integrate information about the processed subscriptions into their *event routing tables*[11]. Additionally, a subscription on its way to the rendezvous broker is routed to all neighbors that previously sent at least one overlapping advertisement (this information is found in the subscription routing table).

The routing of an event message is based on the created event routing tables. It works analogously to the process of using subscription or advertisement forwarding:

- Forward an incoming event message e to all neighbors that previously sent subscriptions fulfilled by e.

The rendezvous broker approach also requires a specialized handling for deregistrations of advertisements and subscriptions. Presenting details about the management of these cases is outside the scope of this dissertation; we again refer interested readers to the original work, for example, the content-based pub-

[8]*Event types*, or just *types*, are an additional concept in content-based pub-sub systems we have not yet introduced. In the type-based approach, event types need to be specified in subscriptions, advertisements, and event messages. The definition of overlap, conforming messages, and matching messages then additionally requires the same type specifications. We refer to Section 4.1 and 7.1 for details.

[9]Several brokers might alternatively be responsible for a particular type, for example, for load balancing purposes, or to allow for the required redundancies in case of broker or network failures.

[10]Note that we refer to a routing table including advertisements as a subscription routing table because this table determines the routing of subscriptions. Conversely, Pietzuch [Pie04] uses the notion of advertisement routing table.

[11]We refer to a routing table including subscriptions as an event routing table because it determines the routing of event messages. Pietzuch [Pie04], however, denotes this concept as a subscription routing table.

sub system HERMES [Pie04], and the topic-based pub-sub systems BAYEUX [ZZJ+01] and SCRIBE [RKCD01].

2.4.5 Assumptions for this Dissertation

As stated previously, our focus does not lie on the development of event routing algorithms. We thus assume the application of existing approaches, and design appropriate routing optimizations as our own contributions.

Event forwarding (Section 2.4.1) is the simplest event routing approach and applicable to applications with constantly changing subscriptions [Müh02] (i.e., registrations and deregistrations). Most scenarios, however, show the pattern of highly-frequent event messages; changes in subscriptions occur regularly but the frequency of changes is marginal compared to the frequency of incoming messages (see Chapter 3). Here the creation of routing tables becomes a beneficial solution that should be exploited to improve the overall system efficiency. We thus do not focus on event forwarding in this dissertation.

When loosening the view of the differences between the remaining routing approaches and only considering their actual event routing processes, all three algorithms merely use the created event routing tables (containing subscriptions as routing entries) to decide on the routing of event messages. An updated, more technical view (compare with Figure 2.6 on page 20) of the routing algorithm using this event routing table is given in Figure 2.11. The subscription-based routing optimization we propose later on (see Chapter 6) manipulates the entries in this table and is thus applicable to all three algorithms. Our advertisement-based optimization (see Chapter 7) requires the registration of advertisements. It can thus be used in conjunction with either the advertisement forwarding (see Section 2.4.3) or the rendezvous broker approach (see Section 2.4.4).

The difference between these two routing solutions incorporating advertisements is the varying number of advertisements and subscriptions distributed among brokers. Advertisement forwarding distributes all advertisements and bases its decisions on distributing subscriptions merely on these advertisements. Rendezvous brokers, on the other hand, forward both advertisements and subscriptions up to the respective rendezvous broker. Subscriptions are distributed even further (but only on their way to the rendezvous broker), based on these advertisements. We do not want to judge which is the preferable solution here, because this decision depends on both the numbers and the

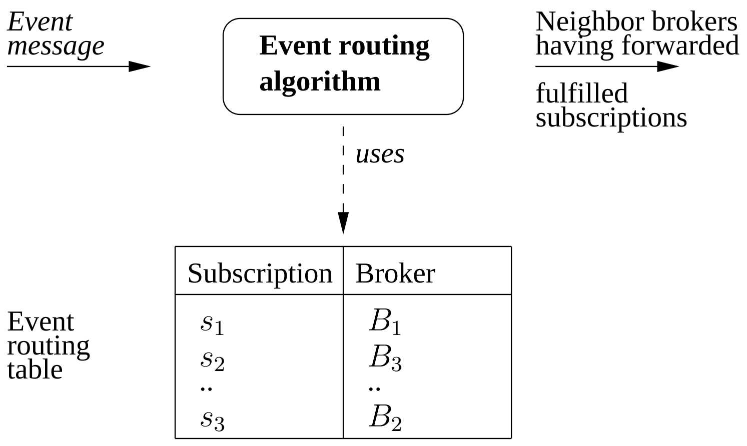


Figure 2.11: Updated view of an event routing algorithm in a broker using the event routing table.

properties of advertisements and subscriptions, which are in turn influenced by the application scenario.

The important point for this dissertation is that our optimization approaches (as well as our filtering algorithm) can be used in combination with all three event routing algorithms. Within this dissertation, we use subscription and advertisement forwarding to exemplify, describe, and analyze our proposals. This choice is primarily based on the independence of these approaches of one crucial parameter existing in the rendezvous broker approach: the placement of rendezvous brokers within the overlay network, strongly affecting the system scalability and efficiency of this approach. Due to our choice, our results show the general effects of our optimizations and lead to universal conclusions without the existence of the additional but essential parameter found in the rendezvous broker approach. A secondary reason for our choice is that the selected routing solutions are easier to implement and do not require specialized rendezvous broker implementations.

Having described our basis for event routing algorithms, we elaborate on existing routing optimizations, their general applicability, and their suitability for Boolean pub-sub systems within the following section.

2.5 Current Routing Optimizations

In this section, we take a closer look at existing routing optimizations for content-based pub-sub systems. As stated in the last section, these routing optimizations are largely applicable to the three main routing approaches:

subscription forwarding, advertisement forwarding, and rendezvous brokers.

Within the following subsections, we introduce the general ideas and concepts of these optimizations, and start an initial analysis of their assumptions and their implications in practice. In Section 2.5.1, we generally elaborate on the concepts and goals of optimizations, and identify several classes of optimizations. The covering optimization is widely applied in content-based pub-sub systems for both advertisements and subscriptions. We present its optimization idea in Section 2.5.2. The next section (Section 2.5.3) then introduces the merging optimization, another approach applicable to both advertisements and subscriptions. Finally, Section 2.5.4 elaborates on subscription summarization, a more recent optimization approach that is based on subscriptions.

2.5.1 Types of Routing Optimizations

The general goal of routing optimizations for content-based pub-sub systems is the improvement of the event routing process with respect to certain quality measures of the system (see Section 2.2). These measures might also implicitly be altered when changing particular system parameters or algorithms (see Figure 2.9, page 27).

Both quality measures we have given in Figure 2.9 (scalability and efficiency) offer the opportunity for optimization. The internal subscription model (as well as the internal advertisement model), on the other hand, has to be supported by the applied optimization approach. (This is the main drawback of current optimizations, being impractical to apply for general Boolean languages, as argued in Sections 2.5.2 to 2.5.4.)

Generally, we can distinguish between *subscription-based* optimizations and *advertisement-based* optimizations. Subscription-based optimizations aim at the manipulation of subscriptions, that is, they alter entries in event routing tables. Advertisement-based optimizations, conversely, attempt to manipulate advertisements, that is, they alter those parts of the system that determine the distribution of subscriptions and are stored in subscription routing tables. We give a graphic view of these two kinds of optimizations in Figure 2.12.

The effect of both kinds of optimizations is a decrease in the memory requirements (to store either subscription base, i.e., event routing tables, or advertisement base, i.e., subscription routing tables). In turn, these memory requirements influence system scalability (see Figure 2.9).

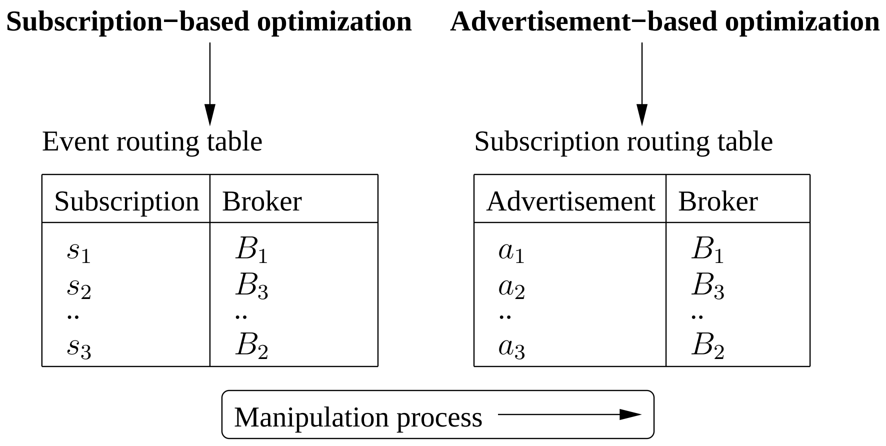


Figure 2.12: Difference between the manipulation in subscription-based and advertisement-based optimizations.

Another way to distinguish optimizations is by their adverse effects. Optimizations are either *non-interfering* or *interfering* optimizations. Representatives of non-interfering optimizations are the covering (see Section 2.5.2) optimization and the perfect merging optimization (a variant of merging, see Section 2.5.3). They aim at influencing one system parameter, the memory requirements, without affecting other system parameters, for example, the network load.

Interfering optimizations, however, influence several parameters at the same time. These optimizations, in general, have certain target parameters to be altered and try to minimally affect other parameters as well. Representatives are the imperfect merging approach (merging variant, see Section 2.5.3), subscription summarization (see Section 2.5.4), and the optimizations we introduce in Chapter 6. They primarily try to decrease the memory requirements, but they secondarily increase the network load as well.

The second routing optimization we propose (see Chapter 7) classifies as advertisement-based interfering optimization. It reduces the memory requirements for subscription routing tables (advertisement base), but it additionally increases the overlapping relationships within the system (accuracy of subscription routing tables). This attribute, in turn, affects the network load for distributing subscriptions and the memory requirements for their storage, that is, the size of event routing tables. We graphically illustrate this way of distinguishing routing optimizations based on their adverse effects in Figure 2.13.

We introduce and analyze existing optimizations within the following subsections.

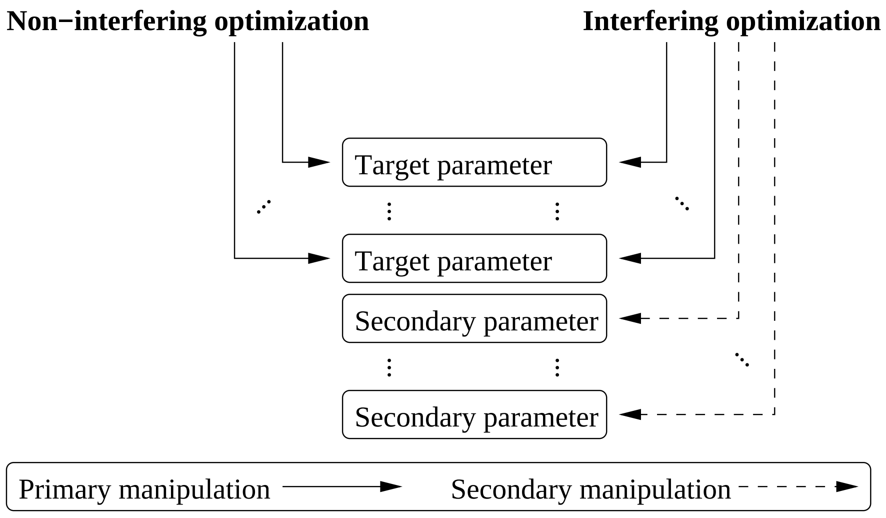


Figure 2.13: Difference between the manipulation in non-interfering and interfering optimizations.

2.5.2 Covering Optimization

Covering aims at removing redundancies among either subscriptions or advertisements, that is, in event routing or subscription routing tables. Its computation is based on event messages potentially fulfilling a subscription and on event messages conforming to an advertisement, respectively.

Let us assume, we have given two subscriptions s_i and s_j. Subscription s_i *covers* subscription s_j if, and only if, $\mathcal{E}(s_i) \supseteq \mathcal{E}(s_j)$. The same definition applies to advertisements: An advertisement a_i *covers* an advertisement a_j if, and only if, $\mathcal{E}(a_i) \supseteq \mathcal{E}(a_j)$. The covering relationship thus describes the inclusion of the set of matching event messages of one subscription s_j (or the set of event messages conforming to an advertisement a_j) in those of another subscription s_i (or in those conforming to another advertisement a_i). Similar definitions of covering have been given in the literature, for example, by Carzaniga and colleagues [CRW01], and by Mühl and Fiege [MF01].

Analyzing the covering relationships among the non-local subscriptions of a broker leads to an optimization potential in combination with the applied routing algorithm: Brokers forward an event message e to neighbors that previously registered at least one subscription fulfilled by e (found in event routing tables). These neighbor brokers, indeed, do not need to know what subscription is fulfilled, because this knowledge is computed by applying the filtering algorithm within this broker. Hence, an intuitive optimization idea to apply to a routing algorithm is to ignore a subscription s_j, forwarded from a neighbor broker B, in routing decisions if B forwarded another subscription s_i that leads to the routing of the same or more messages.

Applying this idea does not affect the routing accuracy and thus the correctness of the system (a more theoretical definition of covering including a formal proof can be found in [Müh02]). Due to this behavior, covering classifies as non-interfering optimization: it only reduces the number of entries in event or subscription routing tables, and thus the memory requirements (for advertisements, one can apply the same ideas based on the overlapping relationships rather than on the matching of event messages).

There is a variant of covering, called subscription subsumption [OJPA06]. Its idea is to exploit the covering relationships among various subscriptions, that is, a subscription might be redundant and does not need to be forwarded because it is covered by a set of other subscriptions. This approach provides an interfering solution for conjunctive subscriptions. Due to the complexity of the subsumption task (being co-NP complete, as shown by Srivastava [Sri93]), [OJPA06] proposes and describes a probabilistic computation algorithm, leading to *false negatives* (subscribers are not notified about matching event messages).

As derived from the previous definitions, one can apply covering as both a subscription-based and an advertisement-based routing optimization. This dual approach is supported by the routing protocols of Siena [CRW01]. However, this work does not analyze the influence of advertisement covering on any system parameter. The same holds for Hermes [Pie04], which only supports type-based advertisements. Subscription covering is applied more widely in content-based pub-sub systems, for example, in the research prototypes Padres [LHJ05], Rebeca [MF01, Müh02], and XRoute [CF03].

The covering optimization, however, shows a number of shortcomings when applied in practice:

General Applicability. The applicability of covering strongly depends on the registered subscriptions and advertisements. If there are no or only a few covering relationships among them, there is no or only a small optimization potential by applying the covering optimization.

The amount of cover that exists generally depends on the application area, including the attribute domains, the typical interests of subscribers, and the typical publication patterns. By and large, covering relationships in practice are based on very restrictive requirements: Even if a subscription s_i describes only one potential event message that is not described by another subscription s_j and vice versa, there is no covering relationship between these two subscriptions (this analogously holds for advertisements when considering the conforming messages).

Overhead for Deregistrations. The efficient support of deregistrations in practice when applying covering is questionable and, to our knowledge, has not been evaluated empirically to date.

Generally, the more optimization potential a particular application scenario leads to, the less efficient the deregistration process becomes. Or, to look at this statement from another viewpoint: the more advantageous the application of covering with respect to efficient and scalable event routing, the more disadvantageous the application of covering with respect to changes in subscription or advertisement base.

The reason for this behavior is the need to recover the removed redundancies (by applying covering) when deregistering. Let us consider the deregistration of a subscription s that covers various subscriptions from the same broker B. To ensure correct event routing, all subscriptions covered by s need to be forwarded by B. Scaling this overhead to the overall network, the deregistration of only one subscription potentially leads to the additional registrations of various subscriptions, affecting both network load and computational load. Furthermore, the covering relationships among these newly registered subscriptions might lead to an even higher overhead. A similar situation occurs when deregistering advertisements. The created load might get even higher in this scenario, because the alteration of the existing overlapping relationships might additionally have effects on subscriptions and even affect their covering relationships.

Internal Subscription or Advertisement Model. The covering optimization has only been applied in combination with conjunctive subscriptions and advertisements so far. Some systems restrict their specifications even further, for example, Rebeca [Müh02] additionally restricts subscriptions and advertisements to contain at most one predicate per attribute, and Hermes [Pie04] only supports type-based advertisements.

Algorithms to efficiently compute the covering relationships among general Boolean subscriptions or advertisements, to our knowledge, do not exist in the pub-sub literature. This is due to the computational complexity of the covering task for this class of expressions. We conclude therefore that covering is impractical to apply for languages more general than conjunctive languages.

Another optimization approach close to covering, called merging, is prevalent for conjunctive subscription and advertisement languages. We elaborate on this proposal in the following subsection.

2.5.3 Merging Optimization

Merging also tries to reduce the number of registered subscriptions and advertisements, and thus the memory requirements for event and subscription routing tables. It does exist in a subscription-based and in an advertisement-based variant. The definition of merging is again founded on the event messages that either match subscriptions or conform to advertisements.

Let us assume we have given a subscription s and a set of subscriptions $\mathcal{S}_i$. Subscription s is called a *merger* of subscription set $\mathcal{S}_i$ if, and only if, $\mathcal{E}(s) \supseteq \bigcup_{s_i \in \mathcal{S}_i} \mathcal{E}(s_i)$. According to the type of set inclusion, one refers to s as a *perfect merger* (for set equality) or as *imperfect merger* (for a proper superset relationship). These two definitions can similarly be applied to advertisements by exchanging s with an advertisement a and $\mathcal{S}_i$ with an advertisement set $\mathcal{A}_i$. Based on these definitions, one can identify two variants of merging: *perfect merging* and *imperfect merging*. Comparable definitions of merging can be found in the literature, for example, by Mühl and Fiege [MF01], and by Li and colleagues [LHJ05].

The general idea of applying merging in practice is as follows: broker components aim at the merging of subscriptions or advertisements that were forwarded by the same neighbor. That is, they decrease the number of routing table entries. When assuming that the created merger requires fewer memory resources, this process reduces the memory requirements of the system.

The creation of a perfect merger only affects the memory requirements but not, for example, the accuracy of event and subscription routing tables. It is thus a non-interfering optimization. When merging subscriptions imperfectly, more event messages (referred to as *false positives*) are forwarded within the broker network, leading to an increased internal network load. This is due to the inaccuracy of event routing tables. The imperfect merging of advertisements leads to an increasing amount of overlap (due to the inaccuracy of subscription routing tables). Hence imperfect merging classifies as interfering optimization.

Due to the merging of only non-local subscriptions and advertisements, the application of any of the two merging approaches does not affect the correctness of the filtering task in content-based pub-sub systems. Local brokers still determine exactly those subscriptions that are fulfilled by an incoming message. Thus, the content-based pub-sub system performs the same notifications as in the un-optimized case.

Subscription-based merging is applied in a range of systems, for example, in the work of Crespo and colleagues [CBGM03], Li and colleagues [LHJ05], and Mühl and Fiege [MF01]. Advertisement-based merging is formally defined, for example, in conjunction with the Rebeca system [Müh02]. However, we are not aware of any evaluation of advertisement-based merging.

The application of imperfect merging poses several unsolved questions, as consistently identified by Mühl [Müh02] and Li and colleagues [LHJ05]: when, what, and how to merge? Li and colleagues [LHJ05] propose an *imperfect degree*, describing the influence of a merger of subscriptions on the number of event messages described by this merger (leading to false positives). They also extend this measure to incorporate registered advertisements.

However, the merging optimization still shows a number of shortcomings in practical application, described as follows:

General Applicability. Although the application of merging does not rely solely on the registered subscriptions and advertisements [Müh02], they still play a significant role in the practical applicability of this optimization. In particular in combination with today's restricted conjunctive subscription and advertisement languages, the potential to represent a perfect merger for a given set of subscriptions or advertisements is very limited.

Hence, the applicability of merging in such systems in practice is questionable if assuming the registration of general subscriptions and advertisements. One can easily imagine the registration of, for example, conjunctive subscriptions that do not allow for the creation of any conjunctive perfect merger. These subscriptions thus cannot be optimized at all by this approach. Generally, the larger the attribute domains, the larger the number of attributes, or the more diverse the registered subscriptions or advertisements, the harder to find perfect conjunctive mergers.

Imperfect merging, on the other hand, is applicable more often. However, its suitability also depends on the registered subscriptions and advertisements, and their definition languages. Generally, the less general a definition language, the more inaccurate the potential mergers. For today's conjunctive languages, the practical suitability of imperfect merging is thus questionable.

Overhead for Deregistrations. Similarly to the covering optimization (see Section 2.5.2), the merging approach leads to a large overhead if subscriptions or advertisements are deregistered.

Let us assume the deregistration of one subscription s_i of a set of subscriptions $\mathcal{S}_i$ (i.e., $s_i \in \mathcal{S}_i$) that is merged into a merger s_j. If s_i is not covered by the other subscriptions in $\mathcal{S}_i$, the merger s_j (and thus event routing table) becomes more inaccurate (or just inaccurate if s_j was a perfect merger). At a certain point of inaccuracy, s_j needs to be replaced by its constituents $s_k \in \mathcal{S}_i \setminus \{s_i\}$ (or one or several new mergers). The deregistration of merger s_j and the registration of its constituents then need to be distributed around the (potentially whole) network.

Merger s_j might, indeed, have been merged in other brokers as well, leading to large inaccuracies and deregistrations of these other mergers. Thus, the deregistration of only one subscription (i.e., s_i) might easily lead to *cascading deregistrations* within the whole network of brokers. But even if this is not the case, one deregistration can cause the additional need to distribute various other registrations and deregistrations, as explained previously. The same effects occur when considering deregistrations of advertisements. The inaccuracy in this case refers to the additional amount of overlap (and the subscription routing table) rather than the additional network load for event routing.

Internal Subscription or Advertisement Model. All current content-based pub-sub systems that support merging are restricted to conjunctive subscriptions and advertisements. The underlying reason for this limitation is the complexity of the general merging task, shown to be NP-hard [CBGM03]. This leads to our conclusion that the support of subscription and advertisement merging in combination with more general languages than conjunctive languages is impractical.

Memory Usage. The optimization goal of merging is to reduce the memory requirements for routing tables and thus to increase system scalability (see Section 2.2). A perfect merger might, however, easily require the same memory resources as its merged constituents. This is particularly the case for diverse subscriptions and advertisements, as argued previously (applicability of merging, see page 54). For imperfect merging, there exists a trade-off between the memory resources for routing tables and their inaccuracy.

Next to merging, targeting dynamic optimization of content-based pub-sub systems, other approaches target the summarization of subscriptions and the distribution of these summaries within the network. We sketch them in the following subsection.

2.5.4 Subscription Summarization

An idea similar to merging is to summarize subscriptions, thus classifying as subscription-based optimization. It was proposed and analyzed by Triantafillou and Economides [TE02, TE04], and by Wang and colleagues [WQV$^+$04].

The overall idea of subscription summarization is to analyze the registered subscriptions and to distribute a summary of these subscriptions to the other brokers in the network. This summary, ideally, requires less memory and allows for a more efficient routing process.

Triantafillou and Economides [TE04] deal with summaries that do not include all subscriptions of all brokers, with the help of additional information carried by each event message. A message includes information about what brokers (i.e., the local subscriptions of these brokers) were already evaluated against the message. Event messages are then forwarded to brokers as long as the subscription summaries of all existing brokers have not been processed.

Wang and colleagues [WQV$^+$04] weaken the concept of local brokers and let the system decide on a broker handling a subscription. This decision is based on the similarity of a new subscription with those subscriptions that are already managed by a broker. Brokers then send summaries of all registered subscriptions to the other brokers in the network. In the event routing process, all brokers with matching subscriptions can thus be calculated.

Both of the described approaches [TE02, WQV$^+$04] use Bloom filters [Blo70] within their subscription summaries. This approach was adopted by Yoneki and Bacon [YB05], supporting events in an XML format and the XPath subscription language.

The created subscription summaries might become imprecise. Hence these proposals categorize as subscription-based interfering optimizations. Although not mentioned in the respective work, there is a potential to apply these ideas to advertisements as well.

The summarization approaches also show a range of shortcomings, similar to the two previously described optimization proposals (covering and merging). They are founded on the same problem areas, and we thus sketch them here:

General Applicability. The applicability of the concept of subscription summaries also depends on the registered subscriptions. For diverse subscriptions, showing little similarity, the benefits of using these approaches are less than when applied for largely similar subscriptions. However, the approach of Triantafillou and Economides [TE04] outperforms the covering optimization when using their experimental setup. Wang and colleagues [WQV$^+$04] conclude by stating that their interfering optimization strongly outperforms the non-interfering variant.

Overhead for Deregistrations. Depending on the frequency of deregistrations, the benefit of a summary that describes all subscriptions of a broker (or several brokers) varies. This is due to the need to update the whole summary in order to propagate changes. Distributing the summaries infrequently results in strong inaccuracies, whereas frequent distribution results in a high network load. One should also keep in mind that updating a summary causes a high computational load due to the need to analyze the relationships among a large number of subscriptions.

Internal Subscription or Advertisement Model. Both summary-based approaches have been proposed for restricted conjunctive subscriptions. The approach of Triantafillou and Economides [TE04] is intertwined with a conjunctive filtering algorithm; it thus cannot be generalized to general Boolean subscriptions. Wang and colleagues [WQV$^+$04] also base their decisions about local brokers for subscriptions on typical properties of mere conjunctive subscriptions, circumventing the utilization of this proposal, for example, for general Boolean subscriptions.

Memory Usage. Experiments by Triantafillou and Economides found that their summarization optimization requires less memory than the covering approach [TE04]. This behavior is due to the exploitation of covering relationships among individual attributes and not only among complete conjunctive subscriptions. However, the reduction in memory requirements still depends on the relationships among the registered subscriptions.

2.5.5 Implications in Practice

Having analyzed existing routing optimizations within the previous subsections, we identify three common problems in recent approaches:

1. Current optimizations are only applicable to restricted conjunctive subscription and advertisement languages. That is, the requirement to support more general subscriptions and advertisements is not fulfilled.

2. The optimization potential of current optimizations depends on the existing relationships among the registered subscriptions or advertisements. That is, current optimizations are only practically applicable in restricted application scenarios.

3. The benefit of current optimizations needs to be paid back when deregistering subscriptions or advertisements. That is, current optimizations assume relatively static subscription patterns.

The development of a routing optimization that targets only one of these problems already constitutes a valuable contribution to current practice. We make such a contribution in this dissertation by presenting two optimization approaches in Chapter 6 and Chapter 7. These approaches not only target one of the identified problems, but tackle all three of these current shortcomings. Based on our novel approaches and their evaluation, we can verify the second part of our central hypothesis (page 6).

Part 1 of this hypothesis regards the unsuitability of canonical conversion for filtering algorithms in general-purpose pub-sub systems. We already hinted, both in this chapter and in Chapter 1, at the problems that occur when current conjunctive solutions are applied to general Boolean expressions. In the following section, we look into these problems in more detail.

2.6 Influences of Canonical Conversion

It is common knowledge that a general Boolean expression, such as included in a subscription or an advertisement, can be rewritten to a canonical form. We refer to this rewriting process as *canonical conversion*. A candidate for a canonical form is the disjunctive normal form [Men97]. For content-based pub-sub systems, it is questionable whether a canonical conversion approach should be taken for subscriptions and advertisements.

Database Management Systems. In database management systems, the restricting clause in database queries is typically internally converted into a canonical form before its execution. Queries are rewritten by database management systems to allow for a common starting point to perform query optimization [JK84]. This optimization is then applied to the conjunctive components of a disjunctive normal form [KMPS94] by employing a selection of predefined conversion rules. Finally, the database management system creates access plans for different ways of processing the query and executes the cheapest plan [JK84].

Pub-Sub Systems. The conversion is already implicitly applied in content-based pub-sub systems if taking the data storage view (see Section 2.1.1, page 15): the transient counterparts to database queries, event messages, are restricted to a canonical property—they are defined as attribute-value pairs with default conjunctive semantics.

Content-based pub-sub systems thus build on the foundation of database management systems with respect to the canonical property of transient data. The conversion of subscriptions and advertisements (stored data), however, does not have an equivalent in database management systems; in these systems, it would correspond to the conversion of all data to a predefined canonical form, such as a flat-file format.

Main Problem: Explosion in Complexity. Our main argument against the practice of converting general Boolean subscriptions or advertisements into disjunctive normal forms is its influence on the memory requirements for their storage (and indexing): a disjunctive normal form, in the worst case, is exponential in size compared to the equivalent general Boolean expression. This implication is consistently acknowledged in the pub-sub area [CCC$^+$01, MFB02].

An exponential increase in size might not occur that often in practice. However, even relatively little increases in complexity already favor the use of general Boolean subscriptions and advertisements over the equivalent canonical form. We show this property throughout this dissertation.

The underlying reason for the inappropriateness of conversion in pub-sub systems is found in (i) the opposite problem definitions in content-based pub-sub and database management systems, and (ii) the opposing application of canonical conversion in these systems. A database management system deals with a small number of *transient and canonically converted* queries at one point in time. Instead, in a content-based pub-sub system, a large number of *stored and canonically converted* subscriptions is registered, and they need to be continuously matched against incoming messages (transient and canonical by definition).

The increase in resources (both memory and computational) required in pub-sub systems when performing conversion is thus, in absolute terms, much higher than in the case of simultaneously executing, for example, a 2-digit number of database queries at one point in time. Additionally, pub-sub systems lack sufficient solutions to optimize subscriptions in general application settings (see Section 2.3), this optimization being the reason for conversion in database management systems.

The increased complexity when converting subscriptions and advertisement affects the main algorithms for pub-sub systems, including the filtering algorithm, the routing algorithm, and the overlapping calculation algorithm.

Consequences: Scalability. The filtering algorithm in pub-sub systems is applied in each individual broker component; its scalability is mainly determined by the memory requirements (space-scalability, see Section 2.2). Canonical conversion increases the size of subscriptions, and thus their memory requirements for storage and indexing. Hence the scalability of individual brokers decreases. Even though there exists some redundancy among converted subscriptions, current filtering algorithms cannot exploit this property (see Section 2.3). A Boolean filtering approach, on the other hand, does not convert in the first place.

The specifications of publishers (advertisements) need to be handled similarly to subscriptions in pub-sub systems [Müh02]. Hence comparable problems and implications regarding scalability arise when converting advertisements.

The applied routing algorithm distributes subscriptions and advertisements as routing entries within the broker network. Thus, if subscriptions and advertisements increase in their overall size, the respective routing tables become larger. Thus, the effects of canonical conversion on central brokers are multiplied in the overall network due to the distribution of subscriptions and advertisements. Besides these effects of canonical conversion on memory requirements, the network load for distributing subscriptions and advertisements increases, also affecting overall system scalability (see Figure 2.9, page 27).

Consequences: Efficiency. The influence of canonical conversion on system efficiency is twofold. On the one hand, filtering algorithms specialized for conjunctive subscriptions exploit the property of only handling conjunctions, and thus do not need to consider the Boolean combination of predicates in subscriptions (see Section 2.3). The same advantageous property holds for the algorithms to calculate the overlap between subscriptions and advertisements.

On the other hand, as an argument against canonical conversion, the size of the problem that needs to be solved by the filtering algorithm or the overlapping calculation algorithm increases. Firstly, conjunctive algorithms need to work on more subscriptions and advertisements (due to their conversion). Secondly, the overall number of predicates within the converted subscriptions or advertisements is much higher.

For the overlapping algorithm, these influences are more severe than for the filtering algorithm. Both subscriptions and advertisements are converted canonically. Hence, both inputs to the algorithm, potentially, are exponential in size, resulting in a multiple explosion of the problem size.

The same overall argument can be applied to the routing task. Routing table entries, on an individual basis, are less complex after conversion than before conversion, that is, routing entries contain fewer predicates that are conjunctively combined per definition. However, the number of routing entries increases exponentially in the worst case, due to canonical conversion.

Conclusions. We give an overview of the identified, twofold influences of canonical conversion on event filtering, event routing, and overlapping task in Figure 2.14. Advantages of conversion are presented on the left-hand side whereas disadvantages are shown on the right-hand side of the figure.

Contemplating the depicted dual effects of canonical conversion instantly raises the question of the benefit of solely conjunctive content-based pub-sub systems. They are clearly advantageous if an application area only requires conjunctive subscriptions and advertisements. However, for scenarios necessitating general Boolean subscriptions and advertisements, this benefit evidently degrades and even transforms into a drawback. Within this dissertation, we

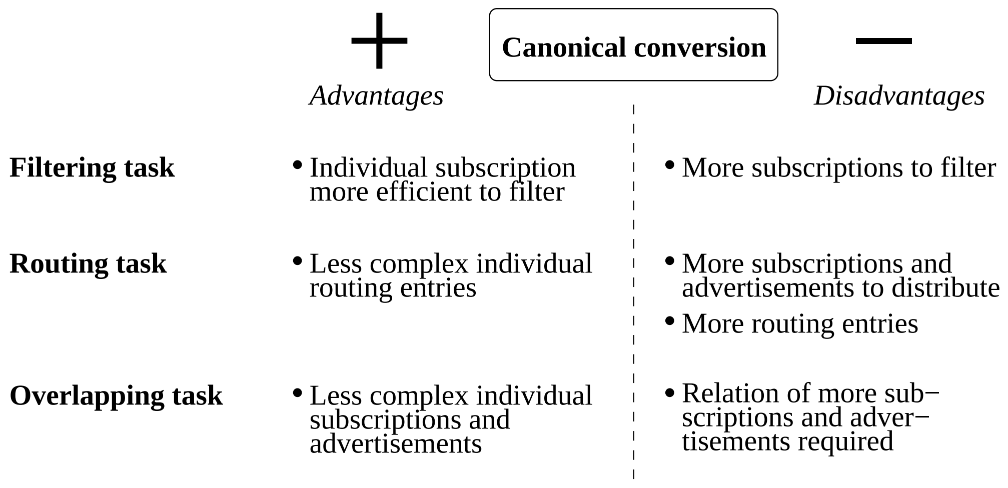

+
Canonical conversion
−
Advantages
Disadvantages
Filtering task
Individual subscription more efficient to filter
More subscriptions to filter
Routing task
Less complex individual routing entries
More subscriptions and advertisements to distribute
More routing entries
Overlapping task
Less complex individual subscriptions and advertisements
Relation of more subscriptions and advertisements required

Figure 2.14: Overview of the influences of canonical conversion on event filtering task, routing task, and overlapping task.

show this behavior, and the general advantages of supporting general Boolean subscriptions and advertisements.

2.7 Summary

Within this chapter, we introduced the general concepts and algorithms for content-based pub-sub systems. Furthermore, we started to analyze recent approaches and to identify their implications.

Content-based pub-sub systems show a range of similarities to database management systems, but there are also fundamental differences between these two kinds of systems. Their most severe dissimilarity is in the vast number of simultaneously registered subscriptions in pub-sub compared to only a moderate number of concurrently processed queries in database management systems. Current content-based pub-sub systems further increase not only the number of registered subscriptions but also the number of registered advertisements due to their sole support of conjunctive expressions. General Boolean subscriptions and advertisements thus need to be converted into disjunctive normal forms to become processable.

This canonical conversion has major influences on the scalability characteristics of content-based pub-sub systems and also on their efficiency properties. Conversion affects the filtering and overlapping calculation tasks in central broker components, as well as the routing tasks within the distributed system. Solutions to these tasks thus experience an explosion in their memory requirements due to the conversion approach taken. The effect of this increased memory use is a degrading of overall system scalability. Regarding efficiency, the influences of conversion are twofold. They decrease the complexity of individual subproblems that need to be solved, but they strongly increase the overall problem size.

The immediate question emerging out of these observations is whether canonical conversion is a suitable operation in content-based pub-sub systems. Within this dissertation, we make a case for the application of content-based pub-sub systems that internally work on general Boolean subscriptions and advertisements. We do so by providing the required filtering, overlapping calculation, and routing solutions supporting these expressions. With the help of our proposals, we show that systems for application scenarios involving general Boolean subscriptions and advertisements can benefit from these more compact expressions: their support leads to an extended system scalability and system efficiency. We introduce one potential application scenario, serving as a running example throughout, in the following chapter.

Chapter 3

Application Scenario: Online Auctions

In this chapter, we introduce an example application scenario for content-based pub-sub systems: online auctions. We gave an initial illustration of some pub-sub functionalities in this scenario in Example 1.1 (page 3). Generally, active notification mechanisms, as offered by pub-sub systems, are highly desirable in online auctions to allow for an efficient dissemination of process-related information [CB02]. We further elaborate on online auctions in general and the benefits of integrating pub-sub mechanisms in Section 3.1.

Subsequently, we analyze the patterns of typical event messages for online auctions (Section 3.2). This is followed by the definition of exemplary subscriptions (Section 3.3) and advertisements (Section 3.4) for this scenario. We use these instances throughout this dissertation to better describe and exemplify our approaches, to apply the developed models, and finally to practically analyze and evaluate our proposals. To further enhance this chapter, we sketch other valuable application scenarios for content-based pub-sub systems in Section 3.5.

The event distributions, and the subscription and advertisement examples we present in the following sections are based on our analysis[1] of auction items on eBay[2]. We restricted this analysis to book auctions, in particular to fiction books offered in the United States. Our results allow for the derivation of a typical event load in online auction settings (as we show in Section 3.2.3).

Combining these typical event distributions with our example subscriptions

[1]The analysis was undertaken on July 8, 2005.
[2]`http://www.ebay.com/`

and advertisements allows for the experimental evaluation of our approaches using this semi-realistic scenario (see Chapter 5 and Chapter 8). This is a valuable advantage over recent evaluations, mostly using purely artificial test settings. The assumptions made to create these artificial workloads are rather strong and hardly ever described in detail. This circumstance does not allow for the repeatability of experiments or comparative evaluations of different approaches by different researchers. This chapter is intended to close this gap, and to describe and provide the foundations of a more realistic test setting.

3.1 Online Auctions

Generally, auctions provide a means to compete for scarce resources or goods. There are several kinds of auctions, but online auctions mostly use a variant of the English open-outcry auction [WWW01]: buyers may place bids on items they are interested in; sellers sell their items to those buyers that submit the highest bids. An auction ends after a fixed amount of time.

On the one hand, online auctions offer advantages for both sellers and buyers, such as the following:

1. Sellers reach a large group of potentially interested customers.
2. Buyers get the opportunity to choose among a great variety of offerings.
3. Items are sold according to the current market price.

On the other hand, several problems arise out of the design of current auctioning platforms, including:

1. Sellers need to present their items appropriately to cope with similar offerings.
2. Buyers have to search through all offered items in order to identify items of interest, leading to an overload in information.
3. Items need to be discovered by manual searching due to the lack of sophisticated notification mechanisms.

These problems do not occur if applying appropriate pub-sub functionalities in online auctions. So let us assume that items would be found automatically if matching a user's subscriptions: the three advantages stated before do still hold. The effect of Advantage 1 gets enhanced even further because it is rather unlikely that buyers miss an item because it is presented inappropriately (which might concern users that usually buy such items cheaply). Additionally, the three problems disappear: Problem 1 is counteracted and solved for the same reason as just described—the content becomes much more important than its presentation. Furthermore, Problems 2 and 3 are automatically tackled by the pub-sub approach, relieving users from the continuous search process. Thus the integration of pub-sub mechanisms into online auctions is a valuable approach, potentially leading to more user satisfaction.

In Section 3.1.3, we describe some of the envisaged pub-sub mechanisms for online auctions and analyze their implications. Beforehand, we describe today's querying functionalities of online auctioning platforms (Section 3.1.1) and currently existing pub-sub support in online auctions (Section 3.1.2).

3.1.1 Existing Querying Functionality

In traditional auctioning platforms, users have to search through all available offerings in order to discover items of interest. These auction sites mostly classify items into categories, allowing for a more effective search process for buyers (due to the automatic suppression of items out of interest).

Commercial auctions offer extended search functionalities, for example, by keyword, end date, price, quantity, or location of the seller. However, users can only combine different criteria using conjunctive semantics. This implies the specification of several queries in order to find items of interest involving non-trivial (i.e., non-conjunctive) specifications (or very broad queries leading to various false positives). Keyword searches in titles and descriptions of auction items, however, allow for more sophisticated functionality, supporting phrase search, and disjunctive and conjunctive semantics as used in information retrieval [WMB99].

In the literature, one can find several efforts to integrate artificial agents into the negotiation process between buyers and sellers, for example, [RWG01]. Such services are particularly worthwhile for commercial users due to their stable interests and their interest in automating the bidding process. Thus the configuration of artificial agents does not change often and might become profitable. Occasional private users, however, rather favor the more traditional approach of visiting auctions and manually bidding for their items. A configuration of artificial agents is not that profitable because each item of interest requires a new specification including the bidding logic.

Furthermore, private users might not be willing to release all of their control to artificial bidding agents. They would rather be involved in the bidding process and, before bidding, analyze the discovered items themselves. If an item has been personally chosen to be worthwhile for bidding (and a maximal price has been determined), artificial agents might be configured, utilizing certain bidding strategies [GFGRAGC98]. Therefore, the exclusive reliance on artificial agents is unrealistic due to the requirement to manually select items and to determine their prices individually. The automatic bidding process, which is the focus of agent approaches, cannot be used to discover items of interest in the first place. We, on the other hand, envisage support for and focus on this discovery process, as described in Section 3.1.3.

Both the artificial and the human approach to buying items can take advantage when using pub-sub functionalities that are provided by auctioning site providers. We describe the currently supported, restricted active mechanisms in the following subsection. Subsequently in Section 3.1.3, we outline those pub-sub functionalities that are required to allow for more effective trading and to eliminate the drawbacks described previously.

3.1.2 Existing Publish-Subscribe Functionality

We can find limited pub-sub functionalities in both commercial auctioning platforms and academic research projects. We describe some of these different systems in the following.

Commercial Auctioning Solutions

Currently, eBay only offers restricted notification functionalities in cases of outbid users and ending auctions. Furthermore, users can ask to be notified about changes in the status of items they are bidding, watching, or selling. Thus, notifications in eBay are purely based on items whose article number is known, that is, items that have been discovered by users using the traditional querying functionality. Consequently, the set of pub-sub mechanisms eBay offers to its users is rather limited.

Yahoo! Auctions[3] supports similar functionalities as eBay but also includes selected, more sophisticated mechanisms: Users get the opportunity to sub-

[3]`http://auctions.yahoo.com/`

scribe to new auctions based on category, seller, or keyword. Hence Yahoo! Auctions supports the discovery of new items by using its pub-sub functions. However, such extended functionalities (compared to eBay) are only supported for new items; for existing auctions, users cannot personalize their subscriptions except by asking to be notified about any changes. Therefore, the active mechanisms provided by Yahoo! Auctions cannot express various user interests properly due to the lack of flexibility to define subscriptions, for example, to receive notifications about items that have a specified price one hour before the end of their auction.

Trade Me[4], the New Zealand auctioning platform, offers notification functionalities similar to Yahoo! Auctions. Items that have been saved to a Watchlist can trigger e-mail notifications either 1, 12, or 24 hours before the end of the auction. Also text messages can be sent to the bidder's mobile phone. Similar to Yahoo! Auctions, new items can be discovered by specifying traditional text queries that are executed on a regular basis, for example, daily. Thus Trade Me does not offer flexible notification mechanisms and items that trigger notifications need to be discovered using traditional querying functionality.

Auctioning Research Projects

IntelliBid, which describes itself as an event-trigger-rule-based auction system [JTS04], allows subscribers to filter on product specifications and categories of items. This functionality improves the mechanisms of commercial solutions; though the generality is still rather limited. Lochner and Wellman [LW04] present a rule-based specification language for online auctions. One might integrate content-based pub-sub mechanisms into online auctions by using this proposal. However, the approach is restricted to conjunctive rules and concentrates on the development of a general and configurable auction service.

In the pub-sub area, the integration of pub-sub functionality into online auctions has been mentioned in the literature, for example, [CW03, LJ04, Müh02]. However, they have failed to analyze the requirements, and the typical workloads and distributions in this application area, which is the focus and goal of this chapter.

[4]`http://www.trademe.co.nz/`

3.1.3 Envisaged Publish-Subscribe Functionality

Having described the state of current solutions, we now illustrate what pub-sub functionalities we envisage in auctioning platforms. Afterwards, we outline the implications of these extensions for users and elaborate on their acceptance.

Enhanced Publish-Subscribe Functionality

As previously mentioned, our goal when integrating pub-sub functionalities into online auctions is to supportively enrich the process of discovering items for potential buyers. We believe that this functionality is highly desirable for private users of online auctions as identified in Section 3.1.1, and it solves the problems discovered in Section 3.1.

We analyzed the existing schema for online book auctions on eBay, in particular for fiction book auctions. This schema allows for the specification of various attributes for book items. We give an overview of these attributes and their domains in Table 3.1. Column 1 shows the name of the attribute; its description and some example values are given in Column 2. The last column specifies the domain for possible attribute values. eBay allows users to specify some of these attributes when querying for items. However, they can only combine their specifications in a conjunctive way, not leaving room for non-trivial queries (see Section 3.1.1) and leading to the drawbacks identified in Section 2.6 (page 58).

When integrating pub-sub mechanisms into auctions, users should be able to restrict all of these attributes in a more flexible way. That is, they should be able to constrain them and to combine these constraints by Boolean operators. This approach allows, for example, the specification of different prices for new and used book copies. We give several examples of simple but non-conjunctive subscriptions in Section 3.3.

Evidently, this support for more complex subscriptions leads to questions regarding user acceptance and user satisfaction. We elaborate on these aspects in the next subsection. However, studying such implications in detail is beyond the scope of this dissertation. This work is left to researchers in the human-computer interaction area.

Acceptance of Enhanced Functionalities

If just searching for items using existing query functionalities, the available conjunctive semantics in current auctioning platforms might be sufficient. Al-

Table 3.1: Overview of attributes for book auctions on eBay.

Attribute	Description or example	Domain or values
`Category`	Category of the book, e.g., `humor`, `poetry`, `fantasy`	Enumeration, 22 values
`Format`	Format of the book, e.g., `hardcover`, `softcover`	Enumeration, 4 values
`Special Attribute`	Special attribute of the book, e.g., `first edition`, `signed`	Enumeration, 3 values
`Condition`	Condition of the book, e.g., `New`, `used`	Enumeration, 2 values
`PayPal`	Specifies whether seller accepts PayPal[5](`yes`/`no`)	Boolean
`Buy It Now`	Specifies whether the book is sold for a fixed price (`yes`/`no`)	Boolean
`Price`	Price of the book	Number, 2 fractional digits
`Auction Title`	Title of the auction	Free text
`Title`	Title of the book	Free text
`Description`	Description of the book	Free text
`Ending Within`	Ending time of the auction, e.g., `1 hour`, `9 days`	Time, up to 10 days
`Language`	Language of the book, e.g., `English`, `French`	Enumeration, 14 values
`Publication Year`	Year of publication	Natural number, up to current year
`Quantity`	Number of books available	Natural number, greater than 1
`Bids`	Number of bids	Natural number, greater than 0

[5]`http://www.paypal.com/`

though, these mechanisms are neither flexible nor general enough for pub-sub functionalities. The conjunctive operator might be the easiest operator to use [YS93], but in pub-sub systems users need to formulate more restrictive subscriptions. This is due to the need to achieve a higher precision[6] in order to avoid annoying, unnecessary notifications. The traditional ranking approach (see, e.g., Maron and Kuhns [MK60]) is not applicable to pub-sub systems. The reason is that the relevance of an auction has to be determined without the knowledge of future items on offer (also previous, already-finished auctions do not necessarily help in determining a rank).

The appreciation of the requirement to formulate more restrictive and precise subscriptions is directly fostered by the benefits for users: the need to fully exploit an existing subscription language will be realized at least in case of large numbers of false positives. Such experiences will lead to more skilled users with an awareness of accurately defined subscriptions. This includes the understanding of various Boolean operators in order to minimize the number of queries as well as the redundancies among queries. Users should thus be willing to invest more time for this definition of long standing subscriptions than for queries in the traditional search process. And these more experienced users are likely to formulate relatively compact Boolean queries [Ros04], that is, queries without strong redundancies.

Additionally, auctioning sites could apply graphical editors, for example, as proposed by Jones and colleagues [JMS99] or Jung [Jun07], to help users in the process of specifying Boolean queries. As already stated, an analysis of such means is out of the focus of this dissertation.

Having described the general application area of online auctions, we proceed with identifying typical event messages and the distributions for this scenario in the next section.

3.2 Event Messages

Let us return to the attributes of online book auctions that are given in Table 3.1. Out of these 15 attributes, we selected and analyzed the distributions of eight main attributes on eBay: `Category`, `Format`, `Special Attribute`[7],

[6]By precision, we refer to the quality measure in information retrieval, as defined in [BYRN99].

[7]Abbreviated by “Attribute” in some figures in the following.

`Condition`, `Buy It Now`, `Price`[8], `Ending Within`[9], and `Bids`.

This approach leads to eight attributes to describe an online book auction. We additionally include the two attributes `Title` and `Author` in our event messages, finally leading to 10 attributes for the event type of book auctions. That is, every event message of this type contains 10 attribute-value pairs (see Section 2.1.1, page 13).

We decided to include these 10 attributes in our analysis due to the requirements of typical subscriptions for online book auctions (see Section 3.3), our personal knowledge and experience with online auctions, and the opportunities to query existing auctions on eBay.

Four of these attributes (`Category`, `Format`, `Special Attribute`, and `Condition`) have enumerations as their attribute domains. We give all possible values for these attributes in Appendix A.1 (page 301). The domains of the remaining attributes are already sufficiently given in Table 3.1. In the following subsections, we analyze the distributions of the values of attributes using nonparametric density estimation [Sil86], and describe how to create event messages based on our findings. We refer to Appendix A.2 (page 303) for details about the distributions of attribute values.

3.2.1 Distribution of Attribute Values

By analyzing eBay[10], we were able to determine the probabilities of all possible combinations of the attributes `Category`, `Format`, `Special Attribute`, and `Condition`. There exist $22 \times 4 \times 3 \times 2 = 528$ combinations of attribute values. The probabilities of book items falling into each of these combinations are given in Appendix A.2. For example, 11.4 percent of all items are characterized as used "Romance" books without special attributes and bound as softcover. No other combination of these four attributes occurs more often than the combination given as example.

For the attribute `Buy It Now`, we analyzed the probability that an item is sold for a fixed price for each category. For example, the highest proportion of fixed-price auctions exists for category "Ancient Literature" and the lowest for category "Poetry" (always compared to the total number of items in these categories).

[8]We use NZ$ as currency in the following, but omit the currency completely in some figures and examples.

[9]Abbreviated by "Ending" in some figures in the following.

[10]The analysis was undertaken on July 8, 2005.

We also evaluated the number of bids depending on the category of items. The largest number of zero-bid items, for example, exists for category “Action, Adventure”; the highest proportional number of items with 10 or more bids is found for category “Pulps” (again, compared to the total number of items per category).

For the attribute `Price`, we evaluated the probability of different price ranges for all categories. For example, 22.2 percent of all “Fantasy” books are sold for more than $10.00; for “Romance” books, this is the case for only 5.9 percent of all items in this category. “Romance” books is also the category having listed the most items for $1.00 or less (38.6 percent).

We additionally analyzed the total number of items per category. For example, “Romance” books constitute the largest proportion overall of auction items (19.2 percent); the smallest proportional number of items is found in category “Ancient Literature”.

The results described in this section were obtained by analyzing all active eBay fiction book auctions on the given date. We also evaluated the distribution of finished auctions on eBay. The probabilities of the different combinations of attributes were approximately the same as in active auctions. Hence the derived results hold for both active and finished book auction items. That is, these distributions generally hold at any given time in the auctioning system.

With the help of these derived distributions, we are able to create event messages that conform to these characteristics of book auction items. We describe how to do this in the next subsection.

3.2.2 Creation of Book Auction Event Messages

We now describe how we use our evaluation results to create event messages that are typical for online book auctions. These messages represent a semi-realistic dataset because we were unable to determine the exact distributions of all possible combinations of the values of all attributes.

That is, our messages contain the real distributions for certain attributes (e.g., `Category`, `Format`, `Special Attribute`, and `Condition`) but are also based on the assumption of simplified dependencies among some attributes (e.g., attribute `Price` only depends on attribute `Category`).

In our analysis, we were unable to derive information about authors (attribute `Author`) and titles (attribute `Title`). This includes, for example, knowledge of the number of unique books per category, the total number of authors, the probability of the same title for different books, and the probability of authors publishing in several categories. Thus for the creation of a workload we have to make certain assumptions, as shown later on.

Creation of Directly Analyzed Attributes

For the determination of the values for the four attributes `Category`, `Format`, `Special Attribute`, and `Condition`, we can directly use the results of our evaluation in Section 3.2.1. That is, we can obtain the values for these four attributes of event messages by choosing a uniformly distributed random number and selecting the respective combination according to the derived probability distribution.

For attribute `Buy It Now`, the probabilities derived in our analysis allow for the calculation of the attribute value based on the already known category. Hence, we can determine whether items should be sold to users with the highest bid (`Buy It Now` = `No`) or for a fixed price (`Buy It Now` = `Yes`).

The calculation of the number of bids (attribute `Bids`) works as follows: based on our findings (see Appendix A.2), we can directly determine the number of bids based on the known category. For 0 to 10 bids, we directly know the probability from our analysis. For more than 10 bids, we derived the probability for intervals of 10, that is, for bids between 11 and 20 up to bids between 51 and 60 (we assume a maximum of 60 bids). Within these intervals, we presume a uniform distribution, that is, all 10 values have the same probability of 0.1.

Our calculation for prices (attribute `Price`) works similarly to the calculation of bids: our analysis led to probabilities of prices based on the known category. Up to \$10.00, we know the distribution of prices in \$1.00 intervals. Within these intervals, we again presume a uniform distribution of the 100 possible values. For higher prices (being relatively rare), we derived the probabilities for intervals of \$10.00 up to the price of \$50.00, for the interval between \$50.01 and \$100.00, and for the interval between \$100.01 and \$1000.00 (we assume \$1000.00 as the maximal price). Our assumption is again a uniform distribution within these intervals.

The remaining three out of our 10 attributes for auction event messages could not be directly evaluated in our analysis of eBay (see Section 3.2.1). We describe their assignment in event messages in the next subsection.

Creation of Other Attributes

We assume a uniform distribution of the termination times of auctions. Hence, the values of attribute `Ending Within` are uniformly distributed up to a maximum of 10 days.

The attributes `Author` and `Title` require more attention than the others because we were unable to derive their distributions on eBay. What we do know, however, is the total number of items for each category (see Section 3.2.1). Furthermore, let us assume the following four parameters: B_{prop} describes the average number of auctions for each unique book title, A_{prop} specifies the average number of books each author has written, p^A_{mult} determines the probability that an author publishes books listed in several categories, and p^T_{mult} states the probability that different books have the same title.

These definitions allow for the following calculations: With the help of parameter B_{prop}, we can determine the number of unique books for each category (dividing the overall number of items per category by B_{prop}). Parameter A_{prop} lets us derive the total number of authors (dividing the number of unique books by A_{prop}).

For each author, we then randomly choose a category (assuming a uniform distribution). With the probability of p^A_{mult}, each author gets associated with a second category. Generally, the probability that an author publishes in n categories is $(p^A_{mult})^{n-1}$.

Finally, for each unique book of a category we choose a random author that is assigned to this category of books. A unique book gets assigned an already used title with probability p^T_{mult}.

To ultimately derive a `Title` of an already determined category, we choose one book that is associated with this category. Because we previously assigned an author to each book, we also successfully determined the value of attribute `Author`.

Using these methods, described in this and the previous subsection, allows us to determine the values for all 10 attributes of event messages of online book auctions. In the next subsection, we describe the meaning of these messages and elaborate on their validity.

Meaning of Created Messages

Using the presented approach to create event messages leads to a semi-realistic dataset. Even though we were not able to determine all properties of the 10 attributes of real-world online book auctions, our dataset is far more accurate than the sole assumption of random distributions within and among these attributes.

We obtained the event distributions by analyzing a snapshot of the items offered at eBay. For attribute `Ending Within`, we assumed a uniform distribution of auction termination times. This assumption might not necessarily be realistic if considering a local eBay site, as the one for the United States, which was chosen for our analysis. However, when assuming an international site, our assumption is realistic and sufficient for our purpose of creating a typical event workload.

An analysis of the bidding times in online auctions, for example, conducted by Hahn [Hah01], reveals the existence of a large proportion of late bidding and, depending on the length of auctions, also early bidding. This typical behavior in bid timing is not explicitly modeled in the event messages that are created by our approach. Although, due to our snapshot analysis, these bidding properties are incorporated into our results: we did not exactly determine the dependency between `Ending Within` and `Bids`. However, the different numbers of bids per category are known and integrated into the generated event messages. The same holds for the prices of items. We do not consider that items that received more bids accumulated a higher price. Instead, the created event messages rather represent the typical values for these attributes for the given categories.

Our snapshot analysis advantageously provides the opportunity to generalize our results, that is, the created event messages: Because messages represent the attributes of all items at a certain point in time, that is, they are ranging from newly inserted to just finished auctions, we can use these event messages to model real events in online auctions. We do not model the history of particular items, but our event messages follow typical distributions of items at any point in time. That is, they also represent events other than the insertion of auction items and the ending of particular auctions.

This fact allows us to model the typical workload in auctions, including the occurrence of real events. These real events, next to the creation and ending of auctions, are bids, that is, users want to pay a higher price for an item than the current highest bidder. Our messages do not exactly model the bidding history for items; nevertheless, they incorporate the numbers of bids and the associated prices at any time, as derived from the analyzed snapshot. We also do not consider the exact times of bids. But, if assuming an international site, the bidding of users is evenly distributed, as modeled in the messages that are created when using our approach.

Having classified the messages generated by our approach, we elaborate on the expected event frequencies in online auction systems in the next section.

3.2.3 Expected Event Frequencies

Event messages in online auctions at least include the insertion of new auction items, the termination of existing auctions, and the bids of users.

In our analysis, 141,602 fiction book items were listed on eBay in the United States. For these items, a total of 38,339 bids existed. Incorporating this number of bids (creating one message each), and the two events of inserting an item and the termination of the respective auction (two messages per item) leads to 321,543 messages in total. Because our analysis included items ending within the next 10 days, this analysis leads to a frequency of approximately 0.37 messages per second.

This result represents a relatively small system throughput. The reasons are obvious: firstly, our analysis only included fiction books. Secondly, it was restricted to items in the United States. Let us thus scale the expected number of event messages to all book auctions hosted at the `eBay.com` site: on February 14, 2006, there was a total of 9,363,317 book auctions on this site. Therefore, the average event throughput to be processed increases to approximately 24.62 events per second.

And now, let us scale the frequency to the total number of items hosted on `eBay.com`: on February 14, 2006, there was a total of 55,771,229 auctions. So the expected event frequency increases to 146.58 events per second if assuming the statistics gained from fiction book auctions. When considering the number of auctions internationally, the system throughput increases even more. Obviously events created by other items than books have a slightly different structure than described in the previous sections.

Let us now assume the pub-sub functionality as an external extension to the auctioning system. To meet our expectations (see Section 3.1.3), for example, to allow users to be notified not earlier than one hour before the end of an auction, the auctioning system needs to create status events at certain time intervals (or the pub-sub extension creates these messages). Assuming these messages to be created in intervals of one hour and considering only fiction books (141,6023 items for 10 days), the additional event throughput to process is 39.33 events per second.

Next to creating the typical event messages in online auctions, we also need to register representative subscriptions in order to produce a characteristic system workload. We proceed with this step in the next section.

3.3 Example Subscription Classes

The definition of characteristic subscriptions for an application scenario is more challenging than the analysis of typical event patterns. The general problem is that an envisaged application does not exist yet and real-world subscriptions cannot be obtained. The random creation of subscriptions is an approach that has mostly been taken to date. We, on the other hand, take a different approach by using predefined classes of subscriptions that are still flexible enough to create a range of settings, for example, involving differing degrees of cover.

We identified three typical classes of subscriptions that would be used in online auctioning systems. We present these classes in this section. *Subscription classes* define the general structure of subscriptions. This structure is represented by the Boolean combination of predicates (see Section 2.1.1, page 12). Predicates are either defined as *fixed predicates*, that is, all subscriptions conforming to a particular class contain these predicates; or they are *variable predicates*, that is, subscriptions of a class get assigned a random predicate value. To represent real world subscriptions, we restrict this randomness of variable predicates to always result in reasonable subscriptions. (We further describe the assumptions we make later on if this is required.)

3.3.1 Definition of Subscription Classes

We depict the structure of Subscription Class 1 in Figure 3.1. All subscription classes that are presented in the following are represented by what we call a *subscription tree*. We properly define subscription trees in Section 4.1.2. In this chapter, we need to notice that the Boolean operators of subscriptions are represented by the inner nodes of these trees. Leaf nodes contain the predicates p of subscriptions; we named these predicates in the figures. Above the root node, we also show the type of the subscription that is represented by the

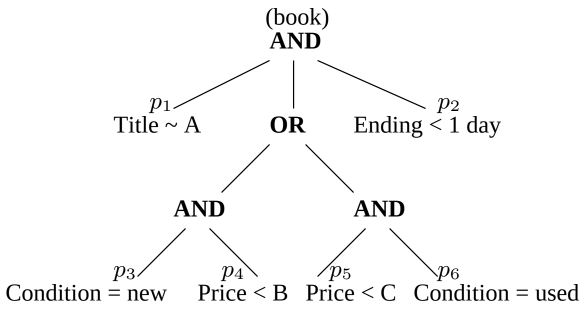


Figure 3.1: Subscription Class 1 including three variable predicates (p_1, p_4, and p_5).

respective subscription tree. This type, though, does not belong to the tree structure.

Definition of Subscription Class 1

Subscription Class 1 (see Figure 3.1), representing an exemplary, typical interest of subscribers, is described as follows:

Subscription Class 1 Users are interested in certain book titles. According to the condition of the copy of the book (either a new or a used copy), they want to pay a different maximal price. To avoid unnecessary notifications, users want to be notified one day before the end of the auction.

In Figure 3.1 the variable predicates of Subscription Class 1 are p_1, p_4, and p_5. Instances of Subscription Class 1 (i.e., actual subscriptions) define particular values for these predicates, for example, `Title` $\sim$ "`Harry Potter`" for p_1, `Price` $<$ `NZ$15.00` for p_4, and `Price` $<$ `NZ$12.00` for p_5 (example subscription s_1). The subscriber would thus like to get notified if a book is on offer whose title contains the phrase "Harry Potter", and which costs less than NZ\$15.00 for a new copy and less than NZ\$12.00 for a used one. The other predicates shown in Figure 3.1 are fixed predicates of this subscription class.

Definition of Subscription Class 2

We illustrate our second class of subscriptions, Subscription Class 2, in Figure 3.2. It is an extension of Subscription Class 1 and its description is as follows:

Subscription Class 2 Again, users are interested in certain book titles and want to be notified one day before an auction ends. The difference to

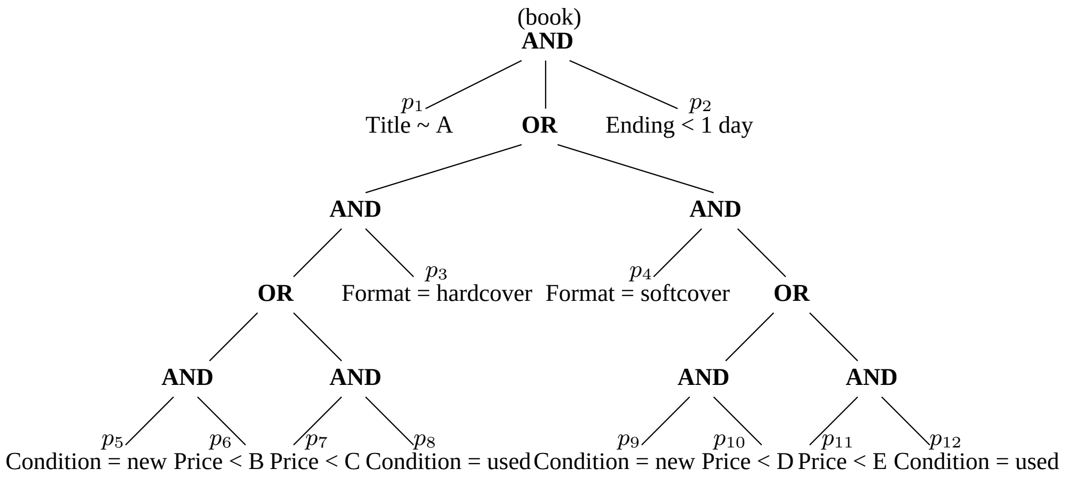


Figure 3.2: Subscription Class 2 including five variable predicates (p_1, p_6, p_7, p_{10}, and p_{11}).

> Subscription Class 1 is that users further distinguish between different formats, that is, between hardcover and softcover books.

Subscription Class 2 contains five variable predicates: p_1, p_6, p_7, p_{10}, and p_{11}. An exemplary assignment of values to these predicates is: `Title` $\sim$ "`Harry Potter`" for p_1, `Price` $<$ `25.00` for p_6, `Price` $<$ `20.00` for p_7, `Price` $<$ `15.00` for p_{10}, and `Price` $<$ `10.00` for p_{11} (example subscription s_2). Using these predicates again describes an interest in books whose title contains the phrase "Harry Potter". Additionally, the price specifications now further depend on the book format. For hardcover books, the subscriber wants to pay at most NZ$25.00 for a new copy and NZ$15.00 for a used book copy. If the book is a softcover, she only wants to pay less than NZ$20.00 for a new and less than NZ$10.00 for a used copy. Subscriptions of this class are thus more restrictive than subscriptions of Subscription Class 1 (assuming the subscription of Subscription Class 2 specifies lower prices for softcovers than for hardcovers, and the subscription of Subscription Class 1 states the same title and the same prices as for hardcovers in the subscription of Class 2).

Definition of Subscription Class 3

The structure of our last class of subscriptions is shown in Figure 3.3. Subscribers specifying a subscription of this class can be characterized as follows:

Subscription Class 3 A collector is interested in books of a certain category but also in books of a particular author. He wants to be notified one hour

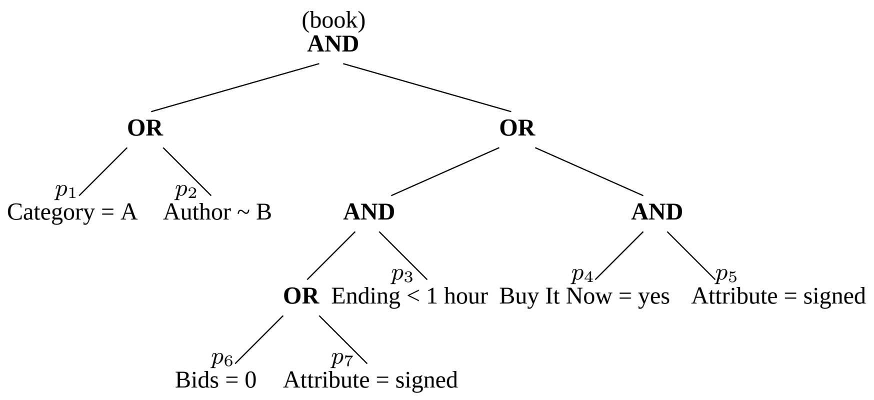


Figure 3.3: Subscription Class 3 including two variable predicates (p_1 and p_2).

> before the end of an auction offering a signed book copy without any bids. Furthermore, he wants notifications about signed books conforming to his interests if the copies can be bought for a fixed price.

This class of subscriptions contains only two variable predicates, p_1 and p_2. These predicates specify the category and the author that are of interest to the collector. An example is `Category` = `Ancient Literature` for p_1 and `Author` $\sim$ "`JK Rowling`" for p_2 (example subscription s_3). These assignments of predicates restrict the author's interests to the two stated values[11].

3.3.2 Properties of Subscription Classes

We give an overview of the properties of our three subscription classes in Table 3.2. This table contains different properties for the three subscription classes in its rows. We show two kinds of properties. The first three rows describe properties of the subscriptions classes in their Boolean form, as depicted in Figures 3.1 to 3.3. We illustrate the number of Boolean operators, the overall number of predicates (fixed plus variable predicates), and the number of variable predicates. The last two rows, however, consider the conversion of these subscription classes to disjunctive normal form (see Section 2.6, page 58). The two properties shown are the number of conjunctive subscriptions that are created out of one Boolean subscription by the conversion and the overall number of predicates in these converted conjunctive subscriptions. The overview

[11]Note that this subscription class is an example showing that a conjunction in natural language (interest in "Ancient Literature" and "JK Rowling") cannot be directly translated into a conjunction in the subscription language. This is a common problem when formulating queries in query languages [YS93].

Table 3.2: Overview of selected properties of our three example subscription classes.

Property	Class 1	Class 2	Class 3
Boolean operators	4	10	6
Overall original predicates	6	12	7
Variable predicates	3	5	2
Converted conjunctions	2	4	6
Overall converted predicates	8	20	18

in this table should give an idea of the increasing complexity of our example subscriptions due to canonical conversion. The following example illustrates this conversion:

Example 3.1 (Conversion of example subscription) *Let us consider example subscription s_1, which was given in Section 3.3.1.*

The conversion to disjunctive normal form results in two conjunctive subscriptions (Row 4 in Table 3.2): Subscription s_{1a} contains predicates p_1, p_2, p_3, and p_4; subscription s_{1b} contains predicates p_1, p_2, p_5, and p_6. Summed up, both conjunctive subscriptions thus contain eight predicates (Row 5 in Table 3.2).

Our subscription classes only describe a selection of the typical interests of subscribers (when populating the variable predicates). However, these classes are sufficient for our purposes of evaluation and are considered as representative within this dissertation.

Even though subscriptions created from these classes inherently contain commonalities, our focus on general-purpose filtering solutions is not impacted upon by this effect: The analyzed filtering algorithms work independently of the commonality pattern. As we describe later on, our subscriptions classes are still flexible enough to create workloads involving, for example, different degrees of cover. We use instances of the presented subscription classes throughout to illustrate our algorithms and approaches, but also within our practical evaluation.

Next to exemplary subscriptions, we define representative advertisement classes. We introduce and describe them in the next section.

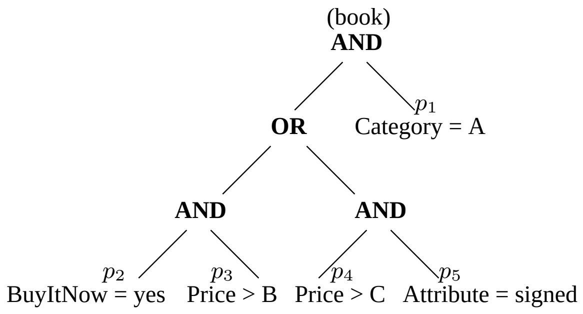


Figure 3.4: Advertisement Class 1 including three variable predicates (p_1, p_3, and p_4).

3.4 Example Advertisement Classes

Analogously to subscription classes, *advertisements classes* define the structure of advertisements, that is, they define the contained predicates and their combination by Boolean operators. There are again two classes of predicates—fixed predicates and variable predicates. We present advertisement classes by *advertisement trees* in the following. These trees have the same definition as subscription trees (see Section 3.3), but they encode advertisements instead of subscriptions.

3.4.1 Definition of Advertisement Classes

Altogether, we identified eight classes of advertisements. We describe and depict these classes in the following. Our goals when developing these advertisement classes were to cover a wide range of possible offerings from publishers and to be consistent with the subscription classes identified before. That is, there should exist some overlap (see Section 2.1, page 11) between instances of these advertisement classes and instances of subscription classes.

Definition of Advertisement Class 1

We depict Advertisement Class 1 in Figure 3.4. Publishers conforming to this class are described as follows:

Advertisement Class 1 A publisher offers books of a particular category. These books are sold for a fixed price (Buy-It-Now item) that is greater than a certain minimal value. The publisher additionally offers signed book copies, also stating a minimal price.

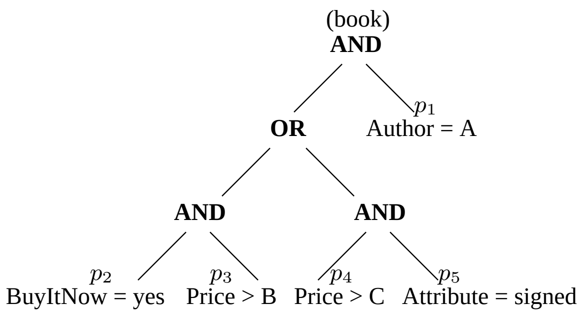


Figure 3.5: Advertisement Class 2 including three variable predicates (p_1, p_3, and p_4).

Instances of this class contain three variable predicates, p_1, p_3, and p_4. An example assignment for these predicates is `Category` $=$ `Poetry` for p_1, `Price` $>$ `15.00` for p_3, and `Price` $>$ `50.00` for p_4 (example advertisement a_1). A publisher specifying these predicates would thus sell books classified as "Poetry" for either a fixed price of more than NZ\$15.00 or for more than NZ\$50.00 for a signed copy.

Definition of Advertisement Class 2

Advertisement Class 2 is similar to Class 1 and illustrated in Figure 3.5. This class describes the following offering from a publishers:

Advertisement Class 2 A publisher is specialized in books of a particular author. She has a range of signed book copies and offers them in online auctions starting with a predefined minimal price. She also wants to sell books for a fixed price (Buy-It-Now items) above a certain threshold.

Similar to Advertisement Class 1, there exist three variable predicates in Advertisement Class 2: p_1, p_3, and p_4. An exemplary instance could define these predicates as follows: `Author` $=$ "`JK Rowling`" for p_1, `Price` $>$ `20.00` for p_3, and `Price` $>$ `100.00` for p_4 (example advertisement a_2). This instance of Advertisement Class 2 describes a publisher who specializes in books from "JK Rowling". The books can be bought for a fixed price of more than NZ\$20.00 (depending on the book). Signed books are sold to the highest bidder and have a starting price of NZ\$100.00.

Definition of Advertisement Class 3

Advertisement Class 3 is visualized in Figure 3.6. Publishers specifying instances of this class could be characterized as follows:

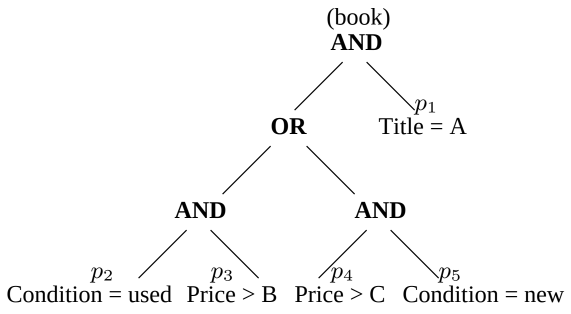


Figure 3.6: Advertisement Class 3 including three variable predicates (p_1, p_3, and p_4).

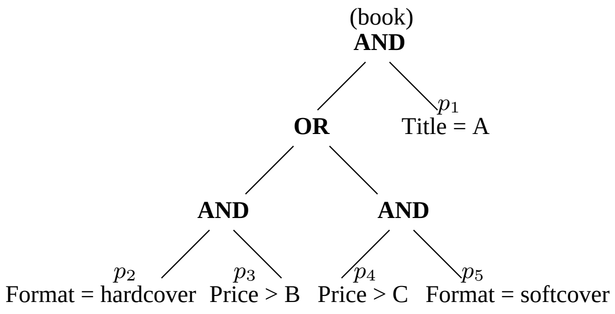


Figure 3.7: Advertisement Class 4 including three variable predicates (p_1, p_3, and p_4).

Advertisement Class 3 A wholesaler has got a stock of books of the same title. Some of these books are slightly damaged and are thus sold as used items. These books are offered for a lower minimum price than the undamaged (i.e., new) items.

Advertisement Class 3 also contains three variable predicates: p_1, p_3, and p_4. A particular publisher could specify these predicates as follows: `Title` = "`Harry Potter and the Goblet of Fire`" for p_1, `Price` > `11.00` for p_3, and `Price` > `14.00` for p_4 (example advertisement a_3). This publisher would have on offer the fourth part of the "Harry Potter" book series for more than NZ$11.00 for a used book copy and more than NZ$14.00 for a new book.

Definition of Advertisement Class 4

Advertisement Class 4 is similar to Class 3, but states varying prices according to the book format instead of the condition of the book copy. It is shown in Figure 3.7 and could be defined as follows:

Advertisement Class 4 A book shop is selling the remainder of its stock of a certain book title via an online auction. The shop owner wants to earn

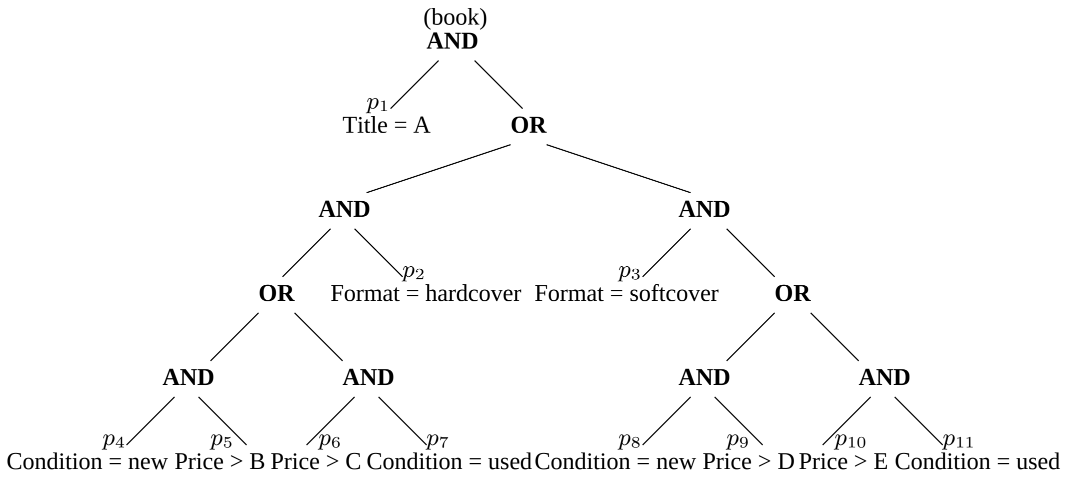


Figure 3.8: Advertisement Class 5 including five variable predicates (p_1, p_5, p_6, p_9, and p_{10}).

at least a particular minimum price according to the format of the book copy. There are hardcover and softcover versions of the book.

There again exist three variable predicates in instances of Advertisement Class 4. They are p_1, p_3, and p_4, and specify book title and minimal prices. An example instance could specify `Title` = "`Harry Potter and the Half-Blood Prince`" for p_1, `Price` > `30.00` for p_3, and `Price` > `20.00` for p_4 (example advertisement a_4). This publisher thus offers the sixth part of the "Harry Potter" series for more than NZ$20.00 for a softcover version and for more than NZ$30.00 for a hardcover version.

Definition of Advertisement Class 5

Advertisement Class 5 combines the two previous classes of advertisements. We graphically illustrate this class in Figure 3.8. Its textual description is as follows:

Advertisement Class 5 A new edition of an academic book has been published. The campus book shop (which is also selling used copies) decides to have a sellout of its large stock of this book. The shop owner wants to use an online auction for the sellout and specifies different minimal prices for softcover and hardcover versions of the book. Additionally, these minimal prices differ for new and used book copies.

Due to the extended specification of this advertisement class, its instances contain five variable predicates: p_1, p_5, p_6, p_9, and p_{10}. We here refrain from

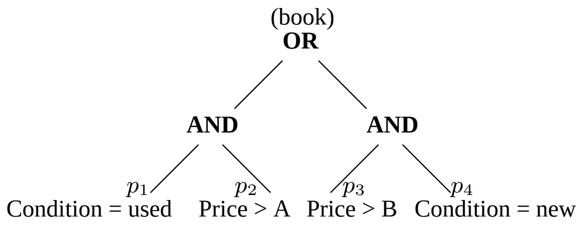


Figure 3.9: Advertisement Class 6 including two variable predicates p_2 and p_3.

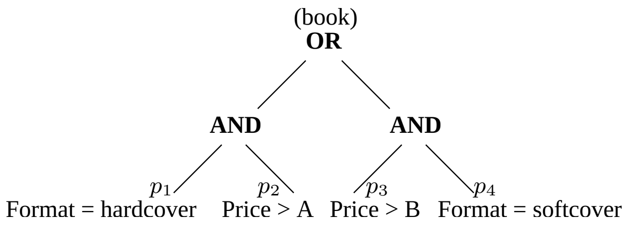


Figure 3.10: Advertisement Class 7 including two variable predicates (p_2 and p_3).

giving an example of an instance of this class. Predicate p_6 should specify a lower price than p_5 and predicate p_{10} a lower price than p_9.

Definition of Advertisement Classes 6 to 8

Our three remaining advertisement classes, Advertisement Classes 6 to 8, are more general versions of Advertisement Classes 3 to 5. We depict Classes 6 to 8 in Figures 3.9 to 3.11. Their descriptions are as follows:

Advertisement Class 6 A publisher offers a broad range of books. She only wants to sell these books if buyers want to pay more than a certain minimal price. This minimal price varies for new and used book copies.

Advertisement Class 7 A book shop has an occasional sellout of various books. The shop owner wants to earn a different minimum amount of money, which only depends on the format of the book.

Advertisement Class 8 To sell a variety of books, a book shop is using an online auctioning system. The minimum prices of the offered books always depend on both the format (hardcover and softcover) and the condition (new and used) of a book.

Advertisement Class 6 (Figure 3.9) and Advertisement Class 7 (Figure 3.10) contain two variable predicates, p_2 and p_3. There are four variable predicates

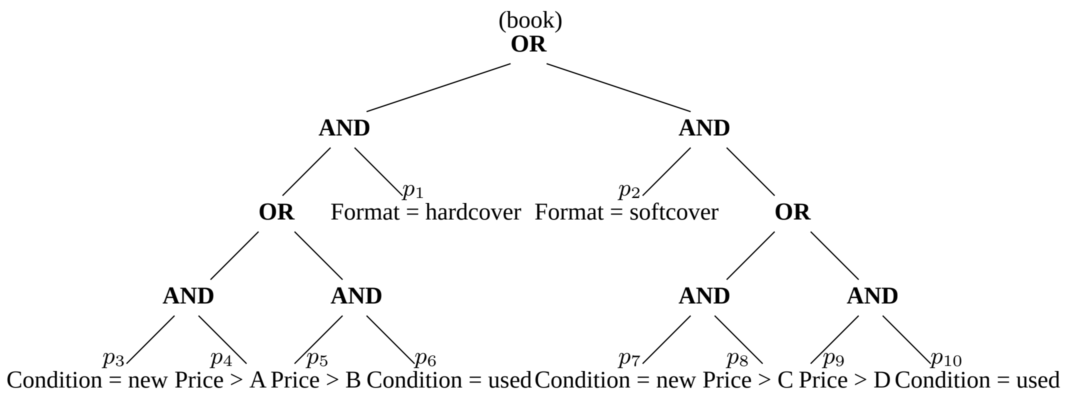


Figure 3.11: Advertisement Class 8 including four variable predicates (p_4, p_5, p_8, and p_9).

for instances of Advertisement Class 8 (Figure 3.11): p_4, p_5, p_8, and p_9. For examples of these classes, we refer to those given for Advertisement Classes 3 to 5 (obviously without the respective predicate on `Title`).

3.4.2 Properties of Advertisement Classes

To give a better overview of the identified classes and the influences of conversion to disjunctive normal form (see Section 2.6), we summarize important properties of our eight advertisement classes in Table 3.3. These properties are shown in the rows of the table for the eight different classes.

Similarly to the identified subscriptions classes, we include three rows that state properties of the Boolean form of the advertisement classes (as shown in Figures 3.4 to 3.11): the number of Boolean operators, the overall number of predicates, and the number of variable predicates. The last two rows of the table describe the effects of conversion of advertisements of these classes to disjunctive normal form: the number of conjunctive advertisements created out of one original advertisement and the overall number of predicates in these converted advertisements.

This description of a wide range of possible advertisements in this section concludes our deliberations on the example application area of online auctions. Similar to the subscription classes, we use instances of the identified advertisement classes to exemplify our approaches and algorithms in due course. Our later experimental evaluation is also based on the findings of this section, that is, on the derived event distributions, and the identified subscription and advertisement classes.

The fact that we identified more advertisement classes than subscription

Table 3.3: Overview of selected properties of our eight example advertisement classes (abbreviated by A1 to A8).

Property	A1	A2	A3	A4	A5	A6	A7	A8
Boolean operators	4	4	4	4	10	3	3	9
Overall original predicates	5	5	5	5	11	4	4	10
Variable predicates	3	3	3	3	5	2	2	4
Converted conjunctions	2	2	2	2	4	2	2	4
Overall converted predicates	6	6	6	6	16	4	4	12

classes is founded in the observation that subscribers typically specify highly selective subscriptions. Otherwise, they would be continuously notified about a variety of items, contradicting the overall pub-sub idea. Applying the identified publisher patterns to subscribers would, in our opinion, lead to unrealistic subscriptions. Subscribers do not want to receive notifications about books in general (Advertisement Classes 6 to 8), about all books of a particular author (Advertisement Class 2), or about all books of a certain category (Advertisement Class 1). We believe subscribers do specify their interest in particular book titles (Subscription Classes 1 to 2), and in more general categories or authors in combination with other, more restrictive predicates (Subscription Class 3).

In the next section, we present other, valuable application areas for content-based pub-sub systems that can benefit from general Boolean subscriptions and advertisements.

3.5 Further Application Scenarios

Besides the online auction application area, various other high-level application scenarios for content-based pub-sub require general Boolean subscriptions and advertisements. We briefly outline two further applications in this section.

3.5.1 Health Care

In the complex application area of health care, pub-sub systems constitute a promising technology, for example, to support patients with chronic conditions [JH05a]. Taking this example deployment situation, the system can be applied to remind patients to take their medication, to manage measurements of various parameters, to inform doctors about critical measurements, and to generally interact with the existing clinical information system including electronic health records.

In the general medical area, the applied management systems and the involved processes are highly complex. It is thus an apparent conclusion that a pure conjunctive pub-sub system might not meet the requirements of this area. For example, taking the task of informing doctors about critical measurements, it mostly does not suffice to solely consider one parameter for such decisions. It is usually a combination of various parameters that leads to critical situations. This naturally introduces different options, that is, disjunctions, into the definition of subscriptions that represent these circumstances. Hence, general Boolean subscriptions are required in such settings.

Clearly, in the health care area, users should not be expected to directly specify subscriptions with a pub-sub system in a given subscription language. Advanced user interfaces need to be employed for this task, as already argued in Section 3.1.3. Furthermore, the support of general Boolean subscriptions is only one requirement of this advanced application area. The detection of complex events (see Section 2.1.3, page 22) and the support of collaboration [JH05b] are further challenging aspects to be addressed. Nevertheless, general Boolean subscriptions are a foundation for these enhancements.

3.5.2 Workflow Management

Workflow management systems, among others, are applied to coordinate and execute processes and activities in distributed environments. Various research in the pub-sub area, for example, by Cugola and colleagues [CNF01], Geppert and Tembros [GT98], and Li and Jacobsen [LJ05], determined workflow management as an application area that naturally relies on event-based mechanisms. Most of the existing work argues for complex event detection to be required in workflow management systems. However, we believe that even at the lowest level of (primitive) event detection more than a mere conjunctive semantics is required by these systems.

Early work on event-based distributed workflow execution [GT98] (event engine Eve) identified 10 attributes for each event in these systems[12]. These

[12]This number of attributes for (primitive) event messages is similar to our finding of 10

attributes describe the occurrence of a particular event and allow for the classification of this occurrence. Event messages in the workflow scenario typically trigger the execution of processes and activities. Subscriptions s (as rules) are associated with, for example, activities (acting as subscribers), and describe those situations that trigger the execution of this activity (i.e., the activity is started if s is fulfilled).

Consider a simple component of a payroll workflow[13]: travel expenses over NZ$1000.00 have to be approved by a manager before they are forwarded to the accounting department, provided the employee is not classified as a consultant. The same approval is required for all business trips of non-consultants that take longer than five days.

The formulation of the rule for this forwarding process to a manager contains at least four predicates: Predicate p_1 describes the type of payment, traveling expenses in this example (e.g., `PaymentType` = `Traveling`). Predicate p_2 restricts the amount to more than NZ$1000.00 (e.g., `Amount` > `NZ$1000.00`). Predicate p_3 defines a duration of more than five days (e.g., `Duration` > `5 days`). Predicate p_4 describes a payment for non-consultants (e.g., `Position` ≠ `Consultant`).

Disjunctively combining p_2 and p_3 results in one subexpression for the final rule. Conjunctively combining this subexpression with p_1 and p_4 finally leads to the overall rule, that is, subscription, for this situation. This general Boolean subscription is then registered with the workflow management system. Using a Boolean expression results in a more space-efficient system than the registration of the two equivalent conjunctive subscriptions in this scenario, as we show in Chapter 5.

3.6 Summary

This chapter gave an impression of the requirement and usefulness of active notification functionalities in the application area of online auctions. Additionally, this chapter provided an analysis of the typical parameters, characteristics, and distributions of event messages, subscriptions, and advertisements in online book auctions.

We based our analyses in this chapter on the existing schema for book auc-

attributes for messages in the auctioning setting (see Section 3.2).

[13]We adopt the general payroll workflow example from PADRES [LJ05].

tions on eBay and on the typical requirements of subscribers and publishers in this area. Our examination led to three important results. The first important finding is the determination of the typical distributions of event messages in the auctioning application scenario. The other two products of our investigations are the formulation of various subscription and advertisement classes. These classes are considered as characteristic throughout this dissertation and are partially used as running examples within the remaining chapters.

These three findings build the foundation of our extensive practical evaluation and, later on, enable us to create typical workloads of pub-sub systems in online auction settings. We use such workloads in our experimental analysis in Chapter 8.

The determination of the typical characteristics of a particular application area is a strong improvement in comparison to existing work. Recent proposals mostly used pure artificial test settings, which, additionally, are not described in detail. Our analysis, however, identifies the attributes of the auctioning application scenario and, in combination with later definitions, allows for the repeatability of our experiments. This, moreover, allows for comparative evaluations with other approaches by different researchers.

To avoid the constriction to online auctions and to keep a general focus on diverse application areas, we also analyzed other implementation scenarios for pub-sub systems. Although the exact determination of typical properties and requirements in these further areas remains future work, we could reason for the need of supporting general Boolean subscriptions in different scenarios.

Chapter 4

Filtering of General Boolean Subscriptions

This chapter proposes a filtering algorithm for general Boolean subscriptions. It is the first filtering approach for this class of subscriptions that applies predicate index structures and thus aims at efficiency aspects. The proposed algorithm categorizes as one-dimensional predicate indexing and individual subscription indexing approach (OP-IS) according to our algorithm classification scheme in Section 2.3.1 (page 29).

In line with our general-purpose requirement, our algorithm represents a generic filtering solution that focuses on the support of general Boolean subscriptions. It extends the conjunctive counting approach (see Section 2.3.3, page 40) to a general Boolean solution. In this dissertation, we do not attempt to optimize our proposal in every possible respect, or to tailor it to certain application scenario specifics[1]. Our focus lies in presenting a general-purpose filtering solution and on additionally introducing a selection of universally applicable optimizations. We integrated this algorithm into our pub-sub prototype, BoP (Boolean publish-subscribe) [BH07].

The structure of this chapter is as follows: In Section 4.1, we start by defining the exact semantics of event messages and general Boolean subscriptions. We need to accurately define these concepts and their exact semantics because a filtering algorithm strongly depends on these definitions. Our algorithm consists of a preprocessing part, performed when subscriptions are registered, and the filtering part itself. We present these constituents of the algorithm in

[1]As an exception, one could state the support of general Boolean subscriptions. This property, however, is not an application scenario specific but required in a broad range of applications, as argued in Chapter 3.

Section 4.2 and Section 4.3, respectively. After outlining some extensions and optimizations to the generic algorithm in Section 4.5, we relate our approach to recent solutions in Section 4.7.

4.1 Event Messages and Subscriptions: Definitions and Semantics

We already introduced the interaction patterns in content-based pub-sub systems and the general directives for using these patterns in Chapter 2. This introduction, however, has been at a high level of abstraction, and the exact definitions of these directives and their meanings are still missing.

Within this section, we thus extend our previous descriptions, and present precise definitions of the notions of event messages (Section 4.1.1) and subscriptions (Section 4.1.2) in BoP. In Chapter 7, we additionally provide a precise definition of general Boolean advertisements.

4.1.1 Event Messages

For our definition of event messages, we assume the existence of *event types* (simply referred to as *types*), as used, for example, in the Cambridge Event Architecture [BBHM95], Hermes [Pie04], and TPS [Eug01]. We define event types as follows:

Definition 4.1 (Event type) *An* event type T *is a set of* event attribute specifications, $T = \{a_1^s, \dots, a_k^s\}$ *in combination with a unique* event type name, T^n. *An event attribute specification,* a^s, *is associated with an* attribute name a^n *(unique within all attribute specifications of an event type, that is,* $\forall a_i^s \in T : \nexists a_j^s \in T \setminus \{a_i^s\} : a_i^n = a_j^n$*), an* attribute domain a^d, *and a set of* Boolean filter functions $a^f = \{f_1 \dots f_k\}$ *of two variables (the first variable of a function* f *is an element in* a^d*; the set of valid second variables is denoted by* f^{op}*):* $a^s = (a^n, a^d, a^f)$. *These Boolean filter functions* f *might be used in subscriptions and advertisements.*

Based on this definition of event types, we define an event message as follows:

Definition 4.2 (Event message) *An* event message e *is a tuple specifying an event type name and a set of* attribute-value pairs*:* $e = (T^n, \mathcal{A})$*, with* $\mathcal{A} = \{(a_1^n, a_1^v), \dots, (a_k^n, a_k^v)\}$.

Each attribute-value pair of an event message e has to specify one of the attribute names that belong to the event type of e (i.e., T) and a value of the respective attribute domain. Every attribute name that is specified by the event type of message e has to be used in exactly one attribute-value pair of e. That is, given an event message $e = (T^n, \mathcal{A})$*, it holds that:*

$$\forall a_i^s \in T : \exists (a_j^n, a_j^v) \in \mathcal{A} : a_i^n = a_j^n \wedge a_j^v \in a_i^d \wedge \\ \nexists (a_k^n, a_k^v) \in \mathcal{A} \setminus \{(a_j^n, a_j^v)\} : a_i^n = a_k^n.$$

This definition of event messages is similar to what is referred to as total messages by Campailla and colleagues [CCC+01], that is, a message defines all attributes of its type. To clarify our previous definitions, in the following we define the example event type T_1 for our online book auction scenario that was introduced in Chapter 3:

Example 4.1 (Event type "book") *The event type* T_1 *describes information about items of online book auctions. It contains 10 attribute specifications and is associated with the name "book", that is,* $T_1^n = \text{book}$*:*

$$\begin{aligned} T_1 = \{&\text{Category}, \text{Format}, \text{Special Attribute}, \text{Condition}, \\ &\text{Buy It Now}, \text{Price}, \text{Ending Within}, \text{Bids}, \text{Title}, \text{Author}\}. \end{aligned}$$

The respective domains of these attributes are given in Table 3.1 (page 71). The supported filter functions are introduced in our example subscription classes (see Section 3.3, page 79) and advertisement classes (see Section 3.4, page 84).

Using this event type specification, we demonstrate an event message e_1 that uses type T_1:

Example 4.2 (Event message of type "book") *Let us assume that a publisher offers a used book entitled "Harry Potter and the Goblet of Fire" as a softcover edition in an auction lasting six hours. The publisher wants to get at least NZ$11.00 for this item. The following event message* e_1 *describes this new auction:*

$$\begin{aligned} e_1 = (&\text{book}, \{(\text{Category}, \text{Fantasy}), (\text{Format}, \text{softcover}), \\ &(\text{Special Attribute}, \text{none}), (\text{Condition}, \text{used}), \\ &(\text{Buy It Now}, \text{no}), (\text{Price}, 11.00), (\text{Ending Within}, 6\ \text{hours}),\end{aligned}$$

$$(\text{Bids}, 0), (\text{Title}, \text{“Harry Potter and the Goblet of Fire”}),$$
$$(\text{Author}, \text{“JK Rowling”})\}).$$

4.1.2 Subscriptions

Subscriptions are also based on event types, as defined in Section 4.1.1. Subscriptions generally describe a filter expression on event messages (see Section 2.1.1, page 12). More precisely, we define a subscription as follows:

Definition 4.3 (Subscription) *A* subscription *s is a tuple that specifies an event type name (see Definition 4.1, page 96) and a* Boolean filter expression*, $s = (T^n, \mathcal{F})$, with $\mathcal{F}$ being a Boolean combination of* predicates *using the operators conjunction, disjunction, and negation. The set of predicates used in $\mathcal{F}$ is denoted by $\mathcal{P}(\mathcal{F})$. Each predicate $p \in \mathcal{P}(\mathcal{F})$ is an* attribute-function-operand triple *(or, more precisely, a triple containing an attribute name, a filter function, and an operand): $p = (a^n, f, op)$.*

Each predicate p_i of the Boolean filter expression $\mathcal{F}$ of subscription s has to specify one of the attribute names (e.g., a^n_j) that belong to the event type of s (i.e., $a^n_j \in T$), a filter function (e.g., f_i) that is included in the set of functions (e.g., a^f_j) specified by this attribute a^s_j (i.e., $f_i \in a^f_j$), and an operand being valid as second variable for this filter function (i.e., f_i). That is, given a subscription $s = (T^n, \mathcal{F})$, it holds that:

$$\forall (a^n_i, f_i, op_i) \in \mathcal{P}(\mathcal{F}) : \exists a^s_j \in T : a^n_j = a^n_i \wedge f_i \in a^f_j \wedge op_i \in f^{op}_i.$$

Our definition of subscriptions is similar to the subscription language classes in [CCC+01] (Simple Subscription Language, Strict Subscription Language, and Default Subscription Language). This similarity to all three subscription languages is due to their property of equal expressivity if applied to total messages, which is our assumption for events (see Section 4.1.1).

Based on our type-based definitions of event messages and subscriptions, we now refine our notion of fulfilled subscriptions and matching event messages to reflect these particular definitions (see Section 2.1.3, page 18 for our original definition):

Definition 4.4 (Fulfilled subscription and matching event message) *A subscription $s_i = (T^n_i, \mathcal{F}_i)$* is fulfilled *by an event message $e_j = (T^n_j, \mathcal{A}_j)$ (equivalent to e_j* matches *s_i) if, and only if:*

1. *Subscription s_i and event e_j specify the same event type, that is, $T_i^n = T_j^n$.*
2. *The Boolean filter expression $\mathcal{F}_i$ evaluates to* true *on event e_j. For this evaluation, each predicate $p_i = (a_i^n, f_i, op_i)$ with $p_i \in \mathcal{P}(\mathcal{F}_i)$ gets assigned the result of the function $f_i(\, a_l^v,\, op_i)$ with $(a_l^n, a_l^v) \in \mathcal{A}_j \wedge a_l^n = a_i^n$. Then, the Boolean combination of these results, stated by $\mathcal{F}_i$, is evaluated.*

For subscriptions s, the set of all matching event messages is denoted by $\mathcal{E}(s)$.

For predicates $p_i = (a_i^n, f_i, op_i)$, the set of matching messages is denoted by $\mathcal{E}(p_i)$. $\mathcal{E}(p_i)$ includes all those messages that result in $f_i(a_l^v, op_i) =$ true *with $(a_l^n, a_l^v) \in \mathcal{A}_j \wedge a_l^n = a_i^n$.*

Using these definitions, subscriptions do not need to contain predicates referring to all attribute specifications defined by their event type. Furthermore, subscriptions might contain several predicates referring to the same attribute specification. The semantics in this scenario are given by the Boolean combination of predicates in the Boolean filter expression. For attribute specifications not referred to by predicates, subscribers do not restrict the attribute value in event messages, that is, they accept all possible values. Whether the incoming event message is matching, solely depends on the stated predicates.

We have already given some example subscriptions in Section 3.3 (page 79). Figures 3.1 to 3.3 depict three example subscription classes we use throughout this dissertation. These classes are represented by subscription trees. In the following subsection, we give details on these tree structures representing the Boolean filter expressions of subscriptions, following this example:

Example 4.3 (Fulfilled subscriptions and matching event messages) *Let us consider message e_1 (see Example 4.2, page 97), and subscriptions s_1, s_2, and s_3 (see Section 3.3, page 79).*

Subscription s_1 is fulfilled by e_1 because both specify event type "book" and the Boolean expression $\mathcal{F}_1$ evaluates to true*: predicates p_1, p_2, p_5, and p_6 (and p_4) get assigned* true*, and hence $\mathcal{F}_1$ results in* true*.*

Subscription s_2 is not fulfilled by e_1. Both specify event type "book" but the Boolean expression $\mathcal{F}_2$ evaluates to false*: predicates p_1, p_2, p_4, p_6, p_7, p_8, p_{10}, and p_{12} get assigned* true*. However, all other predicates get assigned* false*, leading to an overall result of* false *for $\mathcal{F}_2$.*

Subscription s_3 is also not fulfilled by e_1. Although both specify event type "book", the Boolean expression $\mathcal{F}_3$ evaluates to false*: only p_2 and p_6 are fulfilled, resulting in $\mathcal{F}_3$ evaluating to* false*.*

Subscription Trees

Our approach to graphically illustrate the Boolean filter expression of a subscription is to represent the described Boolean combination of predicates by its tree structure. Inner nodes n_i of a subscription tree contain the Boolean operators that are used by the respective subscription; leaf nodes n_l contain the predicates that are used in the filter expression. Note that the event type used in a subscription is not part of its subscription tree (although we stated the type in Figures 3.1 to 3.3 for initial illustration purposes). A subscription tree is comparable with the syntax tree of the Boolean filter expression of a subscription. We denote the set of children of a node n of a subscription tree by $n.children$. (We do not use indices when generally referring to nodes.)

Next to graphically illustrating these filter expressions by their corresponding tree structures, one can use this representation within filtering algorithms for content-based pub-sub systems. We take this approach in BoP. Our internal representation of subscription trees does not store predicates themselves within leaf nodes, but their identifiers. We describe this internal encoding scheme in detail in Section 4.2.2.

Subscription trees are not the only method for representing the Boolean filter structure of subscriptions. Campailla and colleagues [CCC+01] and Li and colleagues [LHJ05] apply Binary Decision Diagrams [JT92]. However, Li and colleagues restrict subscriptions to conjunctions within their proposal. We refer to Section 4.7 for an analysis of the advantages and disadvantages of these alternative representation schemes over the chosen subscription tree approach.

In the following sections, we introduce our filtering approach for general Boolean subscriptions, which is realized in our prototype BoP.

4.2 Preprocessing Step

In this section, we describe the *preprocessing step* of our Boolean filtering algorithm. This initial step is performed if a subscription s is registered with the system. The goal of preprocessing is to index subscription s to allow for an efficient filtering process. Preprocessing includes the syntactical analysis and rewriting of subscriptions (Section 4.2.1), as well as the indexing of predicates and subscriptions themselves (Section 4.2.3). One part of subscription indexing is the internal encoding of subscription trees; we describe this procedure in Section 4.2.2.

Note that preprocessing is merely based on the one subscription that is being registered. It does not create the overhead of shared subscription indexing approaches, requiring a relation of the newly registered subscription and already registered subscriptions (see Section 2.3).

4.2.1 Syntactical Analysis and Rewriting

The first part of the preprocessing step involves two basic rewriting procedures for subscriptions: (i) negation removal, and (ii) operator summarization. These procedures are performed before a subscription is indexed.

Negation Removal

The first rewriting procedure, *negation removal*, is applied to inner nodes of subscription trees. It moves all negation operators in the direction of the leaf nodes by applying De Morgan's laws, that is, negations are pushed down in the subscription tree and integrated into predicates.

This procedure requires the filtering algorithm to always support the negative of a given filter function, for example, equality and inequality, or less than and greater than or equal. In practice, this requirement is straightforward to implement by using the original function for predicate indexing purposes and negating its results. For example, if predicate p_1 of s_3, `Category` = `Ancient Literature`, (see Section 3.3.1, page 81) would be negated due to the shifting down of negations (resulting in `Category` $\neq$ `Ancient Literature`), the filtering algorithm can apply the same index structure as for the original predicate but negate the output for all predicates indexed by this structure. Thus, a system does not need to explicitly consider negations of filter functions other than those already provided.

Operator Summarization

Operator summarization, the second rewriting procedure, analyzes the Boolean operators that are used in subscription trees (representing the filter expression of a subscription) and summarizes consecutive operators of the same kind (i.e., conjunctions and disjunctions, respectively). Proceeding in that way reduces the memory requirements for the encoding of subscription trees, applied later on in the preprocessing step.

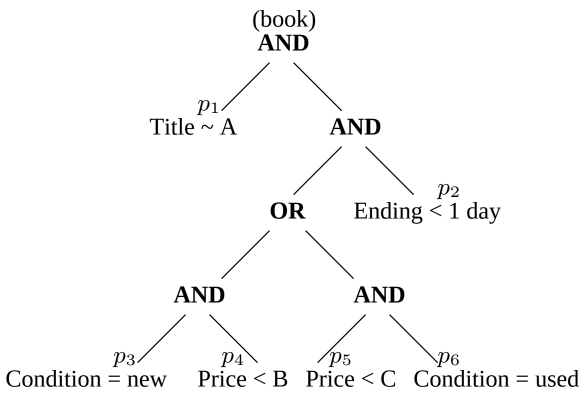


Figure 4.1: Example of Subscription Class 1 before operator summarization.

In our example subscription classes (see Section 3.3, page 79), consecutive operators have already been summarized in their representations as subscription trees in Figures 3.1 to 3.3. The formulation of subscriptions by ordinary system users, however, is likely to result in expanded tree structures. For example, in Figure 4.1 we show a potential formulation of Subscription Class 1 before applying the rewriting procedure. The binary root node and its second child in Figure 4.1 are summarized to a ternary node in Figure 3.1.

Additional Rewriting

Next to these two basic rewriting procedures, one can apply other syntactic rules, for example, to minimize or simplify a given filter expression. Such additional rules can be easily integrated into the algorithm. The semantic rewriting of filter expressions, for example, a worthwhile operation for relational database management systems [GD98], is also a possible extension of BoP. Such extended rewriting steps are beyond the scope of this dissertation.

4.2.2 Encoded Subscription Trees

Having rewritten the original subscription trees, the system performs the indexing of predicates and subscriptions. We describe this second part of the preprocessing step in Section 4.2.3. Before proceeding to these explanations, we discuss the internal encoding of subscription trees.

As stated in Section 4.1.2, there are various ways to internally represent the Boolean filter expression of subscriptions. For the filtering algorithm implemented in BoP, we decided to directly use the introduced subscription trees. Using this straightforward representation approach allows for the general evaluation of our filtering solution and their comparison to conjunctive algorithms independently of low-level realization specifics.

The internal representation of subscription trees uses predicate identifiers $id(p)$ (see Section 4.2.3) in leaf nodes rather than the predicates p themselves. This encoding allows for sharing the occurrence of the same predicates in different subscriptions. In BoP, we use 4-byte unsigned integers as predicate identifiers. A leaf node n_l of a subscription tree is thus represented by 5 bytes: 1 byte denotes n_l as a leaf node; the remaining 4 bytes contain the predicate identifier. (This representation could be optimized to further decrease the memory usage.)

The encoding of inner nodes n_i of subscription trees consists of two parts. The first part (*structural component*) uses 2 bytes to represent the node structure: Byte 1 stores the Boolean operator of the inner node (disjunction or conjunction, negations have been removed, see Section 4.2.1); the number of children is encoded by Byte 2[2]. The second part of the encoding of inner nodes (*functional component*) recursively contains the representation of the children of this node. We exemplify this encoding scheme in the following:

Example 4.4 (Encoding of a subscription tree) *We illustrate an example of the application of our encoding scheme in Figure 4.2. It represents a subscription of Subscription Class 1 (e.g., s_1), given in Figure 3.1 (page 80). Within this example, we use the index of a predicate as its internal predicate identifier (ID in Figure 4.2), that is, for each predicate p_i, it holds $id(p_i) = i$.*

This example subscription requires 38 bytes in total for its encoding. For example, the first 2 bytes describe the structural component of the root node, a conjunction (encoded by a value of 1 in this example) and three children; the remaining 36 bytes describe these three children as the functional component of the root (the first child, a leaf node, requires 5 bytes for its storage.).

We also experimented with a different encoding scheme that stores the width of the internal representation of each child node before the actual node itself [BH05b]. This alternative scheme allows the algorithm to access a child node n_j without having to process all preceding children of the parent of n_j. The approach described here, however, requires the evaluation of all preceding child nodes of n_j in order to be able to access and process n_j (otherwise, the

[2]We thus assume a maximum of 255 child nodes for any node. This assumption aligns with the conjunctive algorithms (see Section 2.3.2, page 31) we use for our later comparison (Chapter 5).

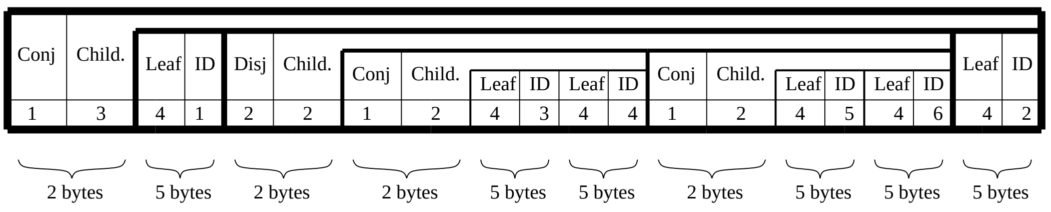


Figure 4.2: Internal encoding of a subscription of Class 1 (cf. Figure 3.1, page 80). The index of a predicate is used as its identifier (ID in this figure), that is, $id(p_i) = i$. We encode a conjunctive node by a value of 1, a disjunctive node by a value of 2, and a leaf node by a value of 4. We also give the memory requirements for the individual parts of the subscription tree.

algorithm cannot determine the memory position of the encoding of n_j).

In BoP, we have chosen the encoding scheme described in this section (cf. Figure 4.2), because it is more space-efficient than the proposal in [BH05b] and shows similar time efficiency properties.

4.2.3 Indexing

Having outlined the encoding scheme for subscription trees, we now proceed with describing the second part of preprocessing: the indexing of subscriptions, including their predicates. After performing this step for a subscription s, this subscription is registered with the system, that is, s is included in the event filtering process, described in Section 4.3.

The overall indexing step consists of two parts, predicate indexing and subscription indexing.

Predicate Indexing

Predicate indexing utilizes one-dimensional index structures. They are specialized with respect to a certain attribute domain and, if necessary, target the efficient implementation of one particular filter function for this domain. For example, equality predicates on integer or float domains could utilize hash tables as index structures; Patricia trees can be used for string domains. Domains of a fixed enumerable size allow for the development of specialized, highly efficient data structures, for example, those described in [AJL02]. The goal of predicate indexing is to provide the filtering algorithm with the means to efficiently determine all predicates that are fulfilled by an incoming event message.

Before indexing, each predicate p is assigned a unique identifier $id(p)$. If subscriptions contain common predicates, that is, predicates p specifying the same attribute name a^n, filter function f, and operand op, these predicates in different subscriptions get assigned the same predicate identifier.

The identification of common predicates is accomplished by a lookup in the respective predicate index. If p is not indexed yet, it is inserted in the index structure and associated with a new predicate identifier $id(p)$. Otherwise, if a predicate is found in the index, the already assigned identifier is used.

Predicate indexing also includes the integration of knowledge about the use of predicates in a *predicate-subscription association table*. This table stores information about the occurrence of each predicate p in subscriptions. For this task, each subscription s gets assigned a unique subscription identifier $id(s)$. The predicate-subscription association table thus maps predicate identifiers to sets of subscription identifiers, that is, it stores $(id(p_i), \{id(s_j), \ldots, id(s_l)\})$ tuples (see Figure 4.4 on page 108 for a graphic illustration). If a subscription s contains the same predicate p several times (e.g., as Subscription Class 2 in Figure 3.2, page 81), p is associated with s more than once in this table. Thus, the subscription set is in fact a multiset. This table is similarly used in the conjunctive counting algorithm (see Section 2.3.3, page 40).

Subscription Indexing

The second part of the indexing step is *subscription indexing*. Using the subscription index structures that are created during this process allows the filtering algorithm to efficiently determine all subscriptions that are fulfilled by an incoming event message (see Section 4.3 for a description of the filter algorithm).

Firstly, the subscription indexing process encodes the filter expression of a subscription s, as described in Section 4.2.2. The memory address $loc(s)$ of the encoded subscription tree of s is then stored in the *subscription location table*, mapping subscription identifiers to memory addresses. This table thus stores $(id(s), loc(s))$ tuples for all indexed subscriptions s (see bottom part of Figure 4.4 for a graphic illustration of this table).

Secondly, subscription indexing calculates a subscription-specific property, $p_{min}(s)$, the *minimal number of fulfilled predicates that is required for a fulfilled subscription s* (shortly referred to as *minimal number of fulfilled predicates*). The value of this property is inserted into the *minimum predicate count vector*, storing $(s, p_{min}(s))$ tuples.

For each subscription s, we can recursively calculate $p_{min}(s)$ by analyzing the structure of its filter expression, encoded in the subscription tree. We show the pseudo code for the calculation of this property for nodes n of subscription trees in Algorithm 1. It works as follows:

- For a leaf node n_l, $p_{min}(n_l)$ is equal to 1 (Line 3 of Algorithm 1).
- For a disjunctive node n_d, $p_{min}(n_d)$ equals the minimum value of $p_{min}(n_j)$ for all children n_j of n_d (Lines 5 to 9).
- For a conjunctive node n_c, $p_{min}(n_c)$ is the sum of the values $p_{min}(n_j)$ of all children n_j of n_c (Lines 11 to 12).

Algorithm 1: Calculation of the minimal number of fulfilled predicates
Input: A node n of a subscription tree
Output: The minimal number of fulfilled predicates $p_{min}(n)$

```
GetMinPredicates(n)
(1)   result ← 0
(2)   if n is a leaf node
(3)      result ← 1
(4)   else if n is a disjunctive node
(5)      foreach c in n.children
(6)         if result = 0
(7)            result ← GetMinPredicates(c)
(8)         else
(9)            result ← min(result, GetMinPredicates(c))
(10)  else if n is a conjunctive node
(11)     foreach c in n.children
(12)        result ← result + GetMinPredicates(c)
(13)  return result
```

The minimal number of fulfilled predicates of a subscription s, $p_{min}(s)$, is equal to the value of this property for the root node n of the filter expression of s, that is, $p_{min}(s) = p_{min}(n)$. We illustrate this calculation in the following example:

Example 4.5 (Calculation of $p_{min}(s)$ for subscription s) *In Figure 4.3, we illustrate the structure of subscriptions of Subscription Class 1 and name all 10 nodes of the subscription tree. The calculation of $p_{min}(s)$ for a subscription s of this class (e.g., s_1) works as follows:*

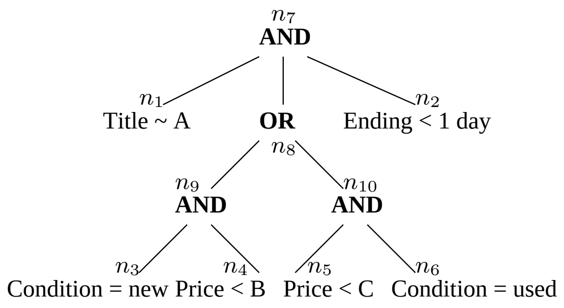


Figure 4.3: Subscription tree of Subscription Class 1 with named nodes.

- *For the leaf nodes, it holds* $p_{min}(n_1) = p_{min}(n_2) = p_{min}(n_3) = p_{min}(n_4) = p_{min}(n_5) = p_{min}(n_6) = 1$.
- *For the conjunctive nodes above the leaf level (*n_9 *and* n_{10}*), it holds* $p_{min}(n_9) = p_{min}(n_3)+p_{min}(n_4) = 2$ *and* $p_{min}(n_{10}) = p_{min}(n_5)+p_{min}(n_6) = 2$.
- *For disjunctive node* n_8*, it holds* $p_{min}(n_8) = \min(p_{min}(n_9), p_{min}(n_{10})) = \min(2, 2) = 2$.
- *For the conjunctive root node* n_7*, it holds* $p_{min}(n_7) = p_{min}(n_1)+p_{min}(n_8)+p_{min}(n_2) = 1 + 2 + 1 = 4$.
- *Finally, for subscription* s *it holds* $p_{min}(s) = p_{min}(n_7) = 4$.

After having performed the subscription indexing process for a subscription s (and the previously described predicate indexing process), BoP considers s in its event filtering algorithm, described in the following section.

4.3 Event Filtering Algorithm

As an extension to the conjunctive counting approach (see Section 2.3.3 on page 40), our general Boolean filtering approach applies a three-step filtering process (predicate matching, candidate subscription matching, and final subscription matching). We illustrate an overview of this process in Figure 4.4.

4.3.1 Predicate Matching

In the *predicate matching* step, the filtering algorithm determines all predicates that are fulfilled by an incoming message. This task is performed by consulting

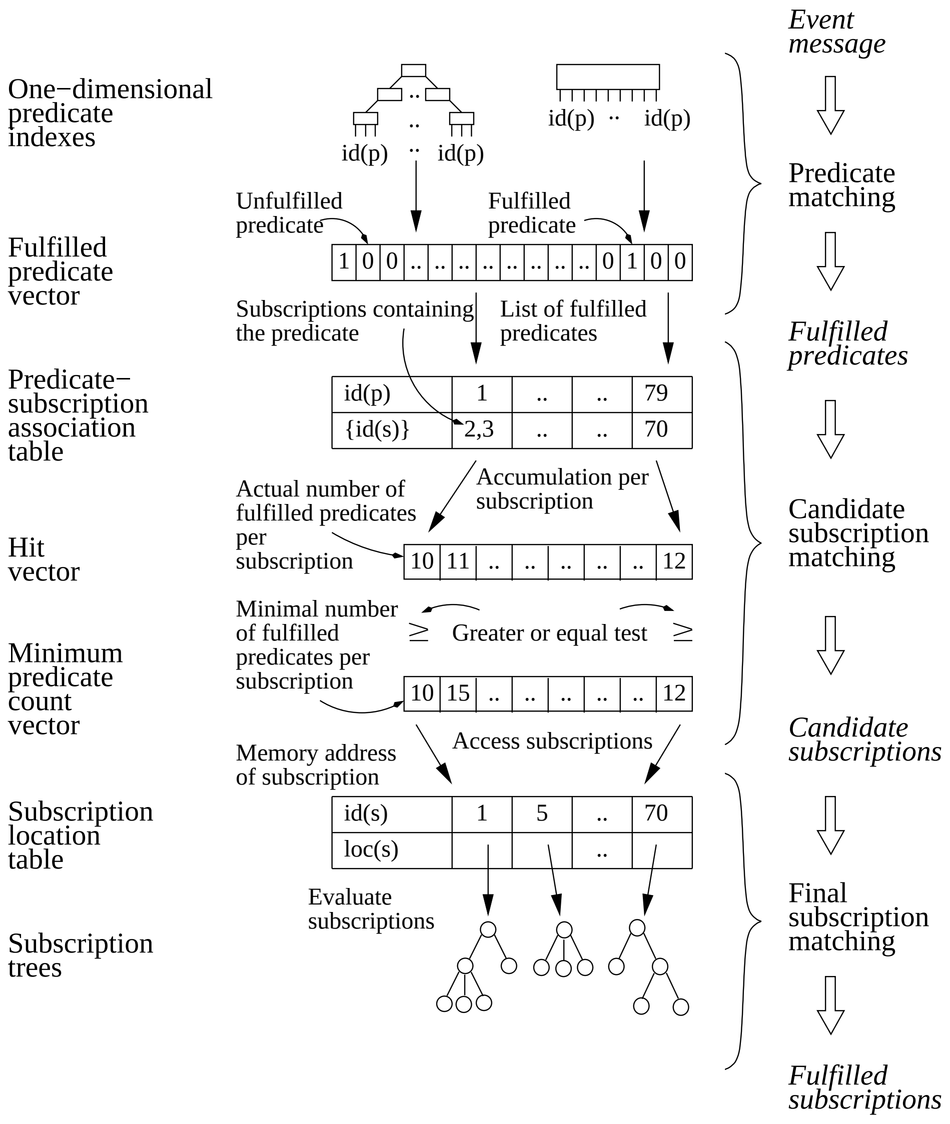

One−dimensional predicate indexes
Fulfilled predicate vector
Predicate−subscription association table
Hit vector
Minimum predicate count vector
Subscription location table
Subscription trees
id(p) .. id(p)
id(p) .. id(p)
Unfulfilled predicate
Fulfilled predicate
1 0 0 .. .. 0 1 0 0
Subscriptions containing the predicate
List of fulfilled predicates
id(p) 1 79
{id(s)} 2,3 70
Actual number of fulfilled predicates per subscription
Accumulation per subscription
10 11 12
Minimal number of fulfilled predicates per subscription
≥ Greater or equal test ≥
10 15 12
Memory address of subscription
Access subscriptions
id(s) 1 5 .. 70
loc(s) ..
Evaluate subscriptions
Event message
Predicate matching
Fulfilled predicates
Candidate subscription matching
Candidate subscriptions
Final subscription matching
Fulfilled subscriptions

Figure 4.4: Overview of predicate matching, candidate subscription matching, and final subscription matching in the Boolean filtering algorithm.

the one-dimensional predicate indexes created in the pre-processing step (see Section 4.2.3): the filtering algorithm evaluates the one-dimensional indexes for all filter functions that are applicable to the attributes of the incoming message. The state of fulfillment of predicates is then recorded in a *fulfilled predicate vector*. Predicate matching is illustrated in the top part of Figure 4.4.

4.3.2 Candidate Subscription Matching

The next step, *candidate subscription matching*, restricts the set of registered subscriptions to a set of *candidate subscriptions* that are potentially fulfilled by the incoming message. The determination of candidate subscriptions is based on the approach taken in the conjunctive counting algorithm (see Section 2.3.3).

The algorithm has provided a set of predicates that is fulfilled by an incoming message. Whether a predicate is fulfilled has been recorded in the fulfilled predicate vector (see Section 4.3.1). Based on this information and the populated predicate-subscription association table (see Figure 4.4 for an illustration and Section 4.2.3 for a description), BoP determines the number of fulfilled predicates per subscription. This task is performed by incrementally increasing a counter in a *hit vector*, containing one 1-byte integer value per subscription. Having processed all fulfilled predicates and evaluated their entries in the predicate-subscription association table, the hit vector states the total number of fulfilled predicates per subscription.

Based on this information, BoP then determines all candidate subscriptions: it compares the value in the hit vector to the value in the minimum predicate count vector (see Figure 4.4 for an illustration and Section 4.2.3 for a description). If the entry in the hit vector shows a value greater than or equal to the entry in the minimum predicate count vector, a candidate subscription is found. The middle part of Figure 4.4 illustrates this candidate subscription matching process.

4.3.3 Final Subscription Matching

Having found the set of candidate subscriptions, the final part of the matching process evaluates this set against the incoming message. Using the subscription location table (see Figure 4.4 for an illustration and Section 4.2.3 for a description), BoP accesses the encoded subscription tree of a candidate. Then, the Boolean structure of the tree is evaluated against the message.

If the filter expression evaluates to *true*, the candidate is a matching subscription. For this evaluation, the filtering algorithm only needs to process the Boolean tree structure of a subscription but not its predicates—the value of the leaf nodes (i.e., the state of fulfillment of predicates) is already known and stored in the fulfilled predicate vector.

We illustrate final subscription matching in the bottom part of Figure 4.4. The following example illustrates the overall filtering process:

Example 4.6 (Filtering of an event message) *Let us consider event message e_1, defined in Example 4.2 (page 97), and the registration of the three* *subscriptions s_1 to s_3 we have given Section 3.3 (page 79). In the following, we refer to the predicates of these subscriptions by p_j^i, stating predicate p_j of subscription s_i.*

The predicate matching step uses the one-dimensional predicate indexes to determine all fulfilled predicates. For the attribute-value pairs of message e_1, these predicates are as follows:

- $\varnothing$ *for* (Category, Fantasy)
- $\{p_4^2\}$ *for* (Format, softcover)
- $\varnothing$ *for* (Special Attribute, none)
- $\{p_6^1, p_8^2, p_{12}^2\}$ *for* (Condition, used)
- $\varnothing$ *for* (Buy It Now, no)
- $\{p_4^1, p_5^1, p_6^2, p_7^2, p_{10}^2\}$ *for* (Price, 11.00)
- $\{p_2^1, p_2^2\}$ *for* (Ending Within, 6 hours)
- $\{p_6^3\}$ *for* (Bids, 0)
- $\{p_1^1, p_1^2\}$ *for* (Title, "Harry Potter and the Goblet of Fire")
- $\{p_2^3\}$ *for* (Author, "JK Rowling")

We now proceed to the candidate subscription matching step: summing up the number of fulfilled predicates for the three subscriptions results in five hits for s_1, eight hits for s_2, and one hit for s_3. The hit vector (using a set notation) is thus $\{5, 8, 2\}$.

In our example, every predicate occurs in only one subscription. This is because we did not identify common predicates when introducing these classes. The filtering algorithm easily retrieves the information about the occurrence of predicates from the predicate-subscription association table. For example, p_3^1 and p_5^2 do internally get assigned the same identifier.

For our three subscriptions, the minimal number of fulfilled predicates is as follows: $p_{min}(s_1) = 4$, $p_{min}(s_2) = 5$, and $p_{min}(s_3) = 3$ (see Example 4.5 on page 106 for a calculation example). The minimum predicate count vector is thus (again using a set notation) $\{4, 5, 3\}$.

The last step of candidate subscription matching identifies candidate subscriptions by comparing the hit and minimum predicate count vector. These *candidates are s_1 ($5 \geq 4$) and s_2 ($8 \geq 5$), but s_3 is not a candidate subscription ($2 \ngeq 3$).*

Final subscription matching finally evaluates the subscription trees of all candidate subscriptions (s_1 and s_2):

Subscription s_1 evaluates to true *and thus is both a candidate subscription and a fulfilled subscription. This is because p_1^1 and p_2^1, as well as p_5^1 and p_6^1 are fulfilled, leading to a subscription tree that evaluates to* true.

However, the other candidate, subscription s_2, evaluates to false. *It is thus a candidate subscription but not a fulfilled subscription. Although p_1^2, p_2^2, and p_4^2 are fulfilled, neither p_9^2 and p_{10}^2, nor p_{11}^2 and p_{12}^2 are fulfilled. Hence, the subscriptions tree of s_2 evaluates to* false.

4.4 Deregistrations

Our filtering algorithm supports a similar deregistration process as the conjunctive counting algorithm. It involves the non-costly removal of subscription information from both predicate and subscription indexes. The algorithm thus does not impose any restrictions with respect to changes in the subscription base.

For the deregistration of a subscription s, the algorithm merely requires its subscription identifier $id(s)$: the entry in the subscription location table can be removed with the help of $id(s)$; the same holds for the minimum predicate count vector. The encoded subscription tree contains all predicate identifiers used by s. This information allows for the removal of the entries in the predicate-subscription association table, but also in the one-dimensional indexes: the predicate-subscription association table states whether a predicate is shared among subscriptions. If the last instance of a predicate is deregistered, it can be removed from predicate indexes. Predicate and subscription identifiers that are not being used anymore should be stored in order to allow for their reuse.

4.5 Algorithm Extensions

One of the design goals of our filtering approach is to provide a solution for general-purpose content-based pub-sub systems. There are also specific extensions and optimizations to the previously presented generic filtering algorithm that significantly improve its performance.

4.5.1 Pure Conjunctive Subscriptions

If subscribers register pure conjunctive subscriptions, BoP handles them with only a little overhead compared to the specialized counting algorithm. In the registration process, BoP analyzes and encodes (see Section 4.2.2) the filter expression of each subscription s. If s is a pure conjunctive subscription, the structural component of the encoded root node of the subscription tree contains an operator identifier that is different from the operator identifier of an ordinary conjunction. The root node is the only inner node in case of conjunctive subscriptions. The filtering algorithm subsequently avoids the evaluation of s in final subscription matching, that is, the evaluation method for the subscription tree just returns *true* without accessing the leaf nodes.

BoP can apply this method, because in a conjunctive subscription s, $p_{min}(s)$ is always equal to the total number of predicates of s. Hence every conjunctive candidate constitutes a fulfilled subscription. The minimal overhead of our filtering approach, in comparison to the counting approach, is to retrieve the memory address of the subscription tree, consulting the subscription location table once. Our experiments confirmed that there is only a marginal overhead of a fraction of a millisecond per filtered event message for processing up to 300,000 subscriptions.

4.5.2 Short-Circuiting

For general Boolean subscriptions, BoP applies a short-circuiting optimization. However, due to the memory-aware encoding scheme of subscription trees (see Section 4.2.2), full short-circuiting can only be applied to root nodes of subscription trees. Inner nodes use partial short-circuiting, that is, nodes are not fully bypassed but only accessed to determine their width in bytes. BoP thus avoids the evaluation of Boolean expressions and the access of the fulfilled predicate vector. For root nodes, on the other hand, BoP applies the full bypass method.

As presented in Section 4.2.2, we also experimented with an alternative encoding scheme that stores the widths of the children of a node and thus allows for full bypassing of any nodes [BH05b]. This alternative scheme requires more memory resources, but led to the same efficiency properties as the applied scheme in empirical experiments.

4.5.3 Order of Children

BoP applies a routing optimization (see Chapter 6) that estimates the selectivity of the nodes of subscription trees. The filtering algorithm uses this information and re-orders the children of a node according to the selectivity estimate. For conjunctions, BoP orders children with increasing selectivity. It is thus more likely to determine a non-fulfilled candidate early in the evaluation process (in final subscription matching). For disjunctions, children are arranged with decreasing values of selectivity estimation. Hence, BoP determines fulfilled candidates early and avoids their further evaluation.

4.5.4 Filtering Shortcut

All approaches in this dissertation work with the subscription or advertisement-forwarding scheme as routing algorithm (see Section 2.4.5, page 46), depending on the application of advertisements. This allows for the implementation of a shortcut optimization to avoid the evaluation of most candidates in final subscription matching. The same shortcut can be applied if subscribers, having various registered subscriptions, only need to be notified about matching events but not about what subscriptions are fulfilled by the message.

BoP uses a hash table (mapping a neighbor broker to a Boolean value) to record whether any non-local subscription that was forwarded by a particular neighbor broker is fulfilled by the incoming event message e. Because e needs to be routed to a neighbor regardless of how many of the forwarded subscriptions are fulfilled, BoP only requires to evaluate the respective candidates until one fulfilled subscription is found. Proceeding in that way avoids the evaluation of the majority of candidate subscriptions in the distributed pub-sub system. The same approach can also be used for subscribers, having properties as described before. An inspiring shortcut optimization was proposed in [CW03] in combination with subscriptions restricted to disjunctive normal form (treating a set of conjunctive subscriptions as one subscription).

4.5.5 Minimal Number of Fulfilled Predicates

The calculation algorithm for the minimal number of fulfilled predicates $p_{min}(s)$ for a subscription s (see Section 4.2.3) only incorporates the syntax of subscription trees. Considering the semantics of subscriptions, however, can lead to a larger value for $p_{min}(s)$. Generally, the higher $p_{min}(s)$ for subscription s, the less frequent s occurs as a candidate subscription in the filtering process. Hence the overall filtering performance is improved for larger values of $p_{min}(s)$.

To exemplify the potential increase of $p_{min}(s)$, let us consider a subscription s of Subscription Class 2, for example, s_2 (see Section 3.3.1, page 80). According to Section 4.2.3, it holds that $p_{min}(s) = 5$. However, one can derive that $p_{min}(s) = 6$ when considering the semantics of s_2 (Figure 3.2, page 81): every fulfilled subscription has to specify either a used or a new book copy. For a new copy, predicates p_5 and p_9 are always fulfilled, whereas predicates p_8 and p_{12} are always fulfilled for used book copies. Either one of these two conditions always holds in practice, leading to two fulfilled predicates. Hence the system still works correctly if increasing $p_{min}(s)$ by one, leading to $p_{min}(s) = 6$. The general goal of this optimization is to incorporate semantic dependencies among predicates into subscriptions.

So far, we have not included this extended semantic analysis of subscriptions into BoP. We plan to do so in the future.

4.5.6 Exploiting Event Types

Event types, on the one hand, define the semantics of subscriptions. On the other hand, one can exploit these types to improve the filtering process: messages can only match subscriptions if they specify the same type (see Definition 4.4, page 98). A filtering algorithm can thus neglect subscriptions of any type other than the one stated by the event message. This restriction is automatically exploited in predicate matching (only predicate indexes of attributes of the respective type are evaluated, confer Section 4.3.1). However, candidate subscription matching (see Section 4.3.2), in the generic way we described previously, offers some optimization potential.

The general idea for this optimization is to compact the hit vector, populated in candidate subscription matching, in order to reduce the number of comparisons that is required to identify candidates. A way of compacting this structure, but still using an efficient array implementation, is to use an advanced handling of subscription identifiers. Firstly, these identifiers contain two parts, one stating the event type and one stating a unique identifier for this type. This allows a specialized hit vector (as an array) to only contain entries for one event type. Secondly, subscription identifiers should not contain holes, that is, the identifier space should be densely populated. This can be achieved by reissuing these identifiers to subscribers, or by adding another level of indirection, that is, internal identifiers differ from those used by subscribers. We plan to fully integrate this extension into BoP in the future.

4.6 Applicability

The presented filtering algorithm constitutes a general Boolean extension of the conjunctive counting algorithm. It inherently shares the advantages of the counting approach with respect to its suitability for general settings.

Firstly, our general Boolean approach does not depend on any patterns concerning the redundancy among predicates. Secondly, the similarity among subscriptions does not have an effect on the internal functioning of the algorithm. This is due to its approach of individually indexing subscriptions, not requiring a particular pattern in this respect. Thirdly, attribute filters only need to be supported by the applied one-dimensional predicate indexes, making subscription indexes independent of this parameter.

Additionally, our general Boolean approach does not show any structural differences in the registration and deregistration process for subscriptions compared to the original counting algorithm. It is thus also suited for applications where changes in the subscription base might occur frequently and not as a rare exception.

One could construct subscriptions s that lead to particularly low values of $p_{min}(s)$. Or, more generally, one could construct subscriptions s whose inherent property $p_{min}(s)$ (stored in the minimum predicate count vector) is greater than or equal to the number of fulfilled predicates for s (stored in the hit vector) for average event messages. Subscription Class 2 is an example of this extreme category of subscriptions. As we will show in the following chapter, even in this case our Boolean filtering algorithm keeps its efficiency properties in comparison to a conjunctive solution. This is due to avoiding the full evaluation of candidate subscriptions of Subscription Class 2 by applying the optimizations presented in Section 4.5.2 (and Section 4.5.3). Our algorithm thus keeps its suitability as a general-purpose solution even in extreme scenarios. The semantic optimization in Section 4.5.5 further improves the algorithm behavior in such extreme settings.

4.7 Related Work

There is no current filtering approach that simultaneously (i) applies predicate indexes to achieve an efficient and scalable filtering process, (ii) supports general Boolean subscriptions, and (iii) is suitable for general application areas. Indexing approaches, for example, [ASS$^+$99, AJL02, FJL$^+$01, GS95, LHJ05, YGM94], are restricted to conjunctive subscriptions, whereas non-indexing solutions, for example, [CCC$^+$01, SA97], support general Boolean subscriptions. We refer to Section 2.3.2 (page 31) for a detailed analysis and description of these algorithms.

The counting algorithm was identified as general-purpose solution (see Section 2.3.2) that offers potential for an extension to support more general than conjunctive subscriptions. We have undertaken this step in this chapter, and proposed a one-dimensional predicate indexing, individual subscription indexing filtering algorithm (OP-IS) for general Boolean subscriptions.

The semantics of subscriptions in BoP is different from conjunctive approaches, for example, Rebeca [Müh02], where all attribute specifications of the event type of a subscription have to be referred to by, at most, one predicate of its conjunctive filter expression. Such restrictions on a subscription language are typically exploited in the algorithms applied in such systems. BoP, however, targets general applications, allows any number of predicates per subscription, and supports a Boolean combination of predicates. This semantics and the generality of the subscription language in BoP are similar to the general Boolean approach in [CCC$^+$01][3].

Another way of storing the filter expression of general Boolean subscriptions (instead of subscription trees) is Binary Decision Diagrams (BDD) [JT92]. Storing individual subscriptions in a form other than subscription trees, such as BDDs, might result in a more space-efficient storage and a more time-efficient evaluation of candidate subscriptions. However, the size of BDDs might get exponential in the number of predicates [Bry86, CCC$^+$01] compared to the linear size of the subscription trees applied in BoP. Generally, using BDDs to represent filter expressions merely means to apply another subscription encoding scheme to BoP. Finding the most promising encoding for general Boolean subscriptions is outside the focus of this dissertation, which is to show the advantages of general Boolean filtering algorithms over conjunctive solutions

[3][CCC$^+$01] restricts the supported filter functions but additionally considers non-total messages.

in general-purpose pub-sub systems.

Approaches that apply shared BDDs for several subscriptions [CCC$^+$01, LHJ05], on the other hand, require the full evaluation of all subscriptions for each incoming message (see Section 2.3.2). Additionally, these approaches are not flexible enough to qualify as general-purpose algorithm solutions. Firstly, the created filtering structures cannot adapt to changes in the characteristics of subscriptions. Secondly, such approaches are designed for scenarios with highly similar subscriptions and can only share subexpressions of subscriptions in this case. They thus degrade to a basic filtering solution if this restrictive pattern does not hold. We refer to Section 2.3.2 for our detailed discussion of these approaches.

4.8 Summary

This chapter described a novel filtering algorithm for general Boolean subscriptions. It is the first algorithm that both supports this class of subscriptions and applies predicate index structures to allow for an efficient filtering process. It is developed as an extension to the generic counting algorithm for conjunctive subscriptions. We integrated the presented filtering approach into BoP, our pub-sub prototype.

The presented filtering algorithm is a general-purpose approach, and we additionally presented a set of optimization methods that are universally applicable. Most of these optimizations are already integrated into BoP. One of them, the filtering shortcut, seamlessly integrates with the routing algorithms that are applied within BoP and thus specifically targets the effective utilization of the proposed filtering approach in distributed settings.

Having presented this filtering algorithm, it remains to investigate whether the algorithm fulfills our design properties: a more space-efficient filtering process than the general-purpose conjunctive solution, achieving at least equal time efficiency properties. We undertake this analysis in the next chapter.

Chapter 5

Boolean or Conjunctive Filtering: A Comparison

In this chapter, we undertake a comparative evaluation between our filtering approach for general Boolean subscriptions, which was introduced in the previous chapter, and the general-purpose conjunctive counting algorithm. Our analysis covers the quality measures system efficiency and system scalability that were introduced in Section 2.2 (page 23). Note that we only evaluate the centralized filtering algorithm in this chapter; our distributed pub-sub prototype is analyzed in Chapter 8.

Our work in this chapter allows us to verify Part 1 of our central hypothesis (page 6):

> *In general-purpose pub-sub systems, a general Boolean filtering approach requires less memory and achieves higher filter efficiency than a conjunctive filtering approach.*

Whereas the time efficiency of filtering algorithms is influenced by the utilized predicates and their combination, the memory requirements of algorithms (and thus the scalability of the central filtering component, see Section 2.2) are largely independent of such scenario specifics. For this purpose, we start this chapter by introducing a characterization framework for subscriptions in Section 5.1. This framework allows us to capture most subscription properties that influence the memory requirements of filtering algorithms applying individual subscription indexing.

Based on this characterization framework, we then describe the memory requirements of the general-purpose counting algorithm and our general Boolean algorithm in Section 5.2 and Section 5.4, respectively. Additionally, we apply the framework to the conjunctive cluster algorithm in Section 5.3. Having modeled the memory use of algorithms in a uniform way, we proceed with comparing them in Section 5.5. After this theoretical part of our analysis, we verify our results for the two general-purpose algorithms in practical experiments, presented in Section 5.6. Finally, we also analyze the filter efficiency of these algorithms in Section 5.7, using our online auction application scenario (Chapter 3).

5.1 Theoretical Subscription Characterization Framework

We now present a theoretical framework [BH05a] that allows for the description of the typical patterns of subscriptions. This framework aims at the evaluation of the memory requirements of filtering algorithms that individually index subscriptions and allows their comparison. Thus our methodology is based on those attributes that affect the memory usage for storing and indexing subscriptions.

In our framework we do not need to model the exact relationships among predicates because all of the filtering algorithms to be analyzed utilize one-dimensional predicate indexes. The memory requirements for these predicate indexes are thus the same. Because our framework ultimately aims at the comparison of the typical memory requirements, the constant memory usage for predicate indexes does not need to be taken into account for this purpose.

Our characterization framework comprises three different classes of parameters:

1. Subscription characterization parameters (Class S)

2. Canonical conversion parameters (Class C)

3. Algorithm-specific parameters (Class A)

Parameter Class S describes the typical structure of (general Boolean) subscriptions, including, for example, the average number of operators and predicates (Section 5.1.1). Parameter Class C characterizes the influence of canonical conversion that is required by conjunctive filtering algorithms. It includes, for example, the number of conjunctive subscriptions that is created by the conversion (Section 5.1.2). Parameter Class A models algorithm-specific implementation details that affect the memory requirements of the analyzed approaches, for example, the size of predicate identifiers (Section 5.1.3).

5.1.1 Subscription Characterization Parameters

Our framework contains six subscription characterization parameters. Two of them can be derived from the combination of some of the other four parameters. One can determine these parameters by analyzing the original Boolean subscription. The parameters are as follows:

Number of Predicates per Subscription $|p|$. This parameter describes the average number of predicates used in a subscription. Considering our representation as subscription trees, $|p|$ states the average number of leaf nodes per subscription.

Number of Boolean Operators per Subscription $|op|$. Subscriptions are defined as general Boolean filter expressions (see Section 4.1.2, page 98). The average number of operators (conjunctions and disjunctions) used in one subscription is denoted by $|op|$.

Proportional Number of Boolean Operators per Subscription op^{prop}. To reduce the number of characterizing parameters, we introduce the proportional parameter op^{prop}. It describes the average number of operators per subscription $|op|$ proportional to the average number of predicates per subscription $|p|$, that is, $op^{prop} = \frac{|op|}{|p|}$.

Number of Subscriptions $|s|$. The number of subscriptions registered with the pub-sub system is referred to as $|s|$.

Number of Unique Predicates $|p_u|$. In order to model predicate commonality, we require the specification of the number of unique predicates that is registered with a pub-sub system. Each unique predicate utilizes a unique predicate identifier and is stored in predicate indexes only once (see Section 4.2.3, page 104). The parameter $|p_u|$ describes the total number of unique predicates registered with the system. It is obviously restricted to the upper bound of $|p| \times |s|$, that is, $|p_u| \leq |p| \times |s|$.

Predicate Commonality p_c. Predicate commonality p_c describes the degree of commonality of predicates, that is, p_c determines whether there are subscriptions that specify the same predicates. We define predicate commonality p_c as the number of shared predicates (occurrences of non-unique predicates) proportional to the overall number of registered predicates, that is, $p_c = 1.0 - \frac{|p_u|}{|p| \times |s|}$. Generally, high predicate commonality occurs in cases of small domain sizes and users with similar interests.

In combination with some algorithm-specific parameters (see Section 5.1.3), these conceptual subscription characterization parameters are sufficient to specify the memory requirements of the previously developed general Boolean filtering algorithm. Conjunctive approaches, however, need to perform canonical conversion to be applicable to general Boolean subscriptions. We require additional parameters, described in the following section, to model this conversion.

5.1.2 Canonical Conversion Parameters

Our subscription characterization framework contains three parameters that describe the influences of canonical conversion. Thus these parameters are only required to model the memory requirements of restricted conjunctive filtering solutions. The third of these parameters can be derived from the former two, as shown in the following descriptions.

Number of Conjunctive Elements After Conversion $|s_s|$. Parameter $|s_s|$ describes the average number of conjunctive subscriptions created by the canonical conversion of one original (i.e., general Boolean) subscription. That is, it describes how many conjunctive elements (individual subscriptions) are combined by the disjunction in the created disjunctive normal form.

Number of Conjunctive Elements per Predicate After Conversion $|s_p|$. A conversion to disjunctive normal form implies that predicates of a general Boolean subscription participate in several conjunctive elements. Parameter $|s_p|$ describes in how many conjunctive elements a predicate from the original subscription occurs on average. (In case of predicate commonality, each occurrence of the predicate is treated separately.)

Proportional Number of Conjunctive Elements per Predicate After Conversion s^{prop}. To reduce the number of parameters in the later comparison of memory requirements, we introduce the proportional notion s^{prop}. It denotes the average number of conjunctive subscriptions per original predicate (i.e., $|s_p|$) proportional to the total number of conjunctive subscriptions created by the canonical conversion (i.e., $|s_s|$), that is, $s^{prop} = \frac{|s_p|}{|s_s|}$.

These three parameters are sufficient to model the effects of canonical conversion of subscriptions. In combination with the algorithm-specific parameters we introduce in the next section, one can model the memory requirements of conjunctive algorithms (see Sections 5.2 to 5.3). Obviously, our general Boolean solution does not depend on these conversion-related parameters, but only on the parameters defined in the previous and the next section.

5.1.3 Algorithm-Specific Parameters

The parameters introduced in the two previous sections describe the general subscription structure and the influence of canonical conversion independently of the actually applied filtering approach. The implementation of particular algorithms, however, requires the incorporation of algorithm-specific parameters into our subscription characterization framework. The former two of the following parameters are required by a wide range of algorithms, whereas the latter two are particularly required by the approaches we use in our later analysis.

Width of Subscription Identifiers $w(s)$. Parameter $w(s)$ describes the width of subscription identifiers in bytes. These identifiers are used within all three of the analyzed algorithms later on. If using 32-bit unsigned integers as identifiers, it holds $w(s) = 4$.

Width of Predicate Identifiers $w(p)$. Predicates also have to be uniquely identifiable by the algorithms we analyze. Parameter $w(p)$ specifies the width of predicate identifiers in bytes. Again, using 32-bit unsigned integers as identifiers leads to $w(p) = 4$.

Width of Subscription Locations $w(l)$. In our Boolean filtering approach, we store subscriptions as subscription trees (see Section 4.2.2, page 102). The memory positions of these trees are referenced by a subscription location table. The width of such a location reference in bytes is denoted by $w(l)$. If utilizing standard memory pointers on 32-bit machines, it holds $w(l) = 4$.

Table 5.1: Overview of the parameters of our subscription characterization framework (Class S–subscription characterization parameters, Class C–canonical conversion parameters, and Class A–algorithm-specific parameters).

Symbol	Parameter Name (Calculation)	Class
$\lvert p \rvert$	Number of predicates per subscription	S
$\lvert op \rvert$	Number of Boolean operators per subscription	S
op^{prop}	Proportional number of Boolean operators per subscription ($op^{prop} = \frac{\lvert op \rvert}{\lvert p \rvert}$)	S
$\lvert s \rvert$	Number of subscriptions	S
$\lvert p_u \rvert$	Number of unique predicates	S
p_c	Predicate commonality ($p_c = 1.0 - \frac{\lvert p_u \rvert}{\lvert p \rvert \lvert s \rvert}$)	S
$\lvert s_s \rvert$	Number of conjunctive elements after conversion	C
$\lvert s_p \rvert$	Number of conjunctive elements per predicate after conversion	C
s^{prop}	Proportional number of conjunctive elements per predicate after conversion ($s^{prop} = \frac{\lvert s_p \rvert}{\lvert s_s \rvert}$)	C
$w(s)$	Width of subscription identifiers	A
$w(p)$	Width of predicate identifiers	A
$w(l)$	Width of subscription locations	A
$w(c)$	Width of cluster references	A

Width of Cluster References $w(c)$. The cluster algorithm clusters subscriptions according to common access predicates (see Section 2.3.2, page 37). The width of a reference to such a cluster in bytes is stated by $w(c)$. Using standard memory pointers on 32-bit machines leads to $w(c) = 4$.

Although one cannot model all details of certain algorithm implementations with these parameters, they allow for the derivation of the main memory requirements of the three algorithms we analyze, as shown in Sections 5.2 to 5.4. Before proceeding to these algorithm analyses, we characterize subscriptions of our three subscription classes with the help of our framework. To allow for a better overview of all 13 parameters, we present a compact summary in Table 5.1.

Table 5.2: Overview of a selection of parameters of our subscription characterization framework for Subscription Classes 1 to 3 (see Figures 3.1 to 3.3, page 80).

Parameter	Class 1	Class 2	Class 3
$\|p\|$	6	12	7
$\|op\|$	4	10	6
op^{prop}	$\frac{4}{6} \approx 0.667$	$\frac{10}{12} \approx 0.833$	$\frac{6}{7} \approx 0.857$
$\|s_s\|$	2	4	6
$\|s_p\|$	$\frac{2\times2+4\times1}{6} \approx 1.333$	$\frac{2\times4+2\times2+8\times1}{12} \approx 1.667$	$\frac{2\times3+4+4\times2}{7} \approx 2.571$
s^{prop}	$\frac{1.333}{2} \approx 0.667$	$\frac{1.667}{4} \approx 0.417$	$\frac{2.571}{6} \approx 0.429$

5.1.4 Characteristics of Example Subscription Classes

We now characterize our three example subscription classes (see Section 3.3, page 79) with the help of the introduced subscription characterization framework. We only state those parameters that directly influence the development of the memory requirements of the analyzed algorithms (Sections 5.2 to 5.4). Table 5.2 gives a detailed overview of these parameters of our framework. In the rows of the table, we present the parameters of our framework; columns contain the different subscription classes. In Section 5.5.3, we use the same parameters to determine whether a conjunctive or a general Boolean filtering algorithm requires less memory for the indexing of subscriptions of these classes.

The following example illustrates the calculation of these parameters for Subscription Class 2 (see Column 3 of Table 5.2):

Example 5.1 (Characterization of Subscription Class 2) *Class 2 (Figure 3.2, page 81) contains 12 predicates, p_1 to p_{12}, leading to $|p| = 12$. In Section 3.3.1 (page 79), we have given an example subscription of this class, subscription s_2. The 12 predicates are combined by 10 Boolean operators, seven conjunctions and three disjunctions. Hence, it holds $|op| = 10$ for this subscription class. Combining these two subscription characterization parameters leads to $op^{prop} = \frac{|op|}{|p|} = \frac{10}{12} \approx 0.833$.*

Performing a canonical conversion leads to four conjunctive subscriptions for this class, $s_{2a} = (\text{book}, \mathcal{F}_{2a})$ to $s_{2d} = (\text{book}, \mathcal{F}_{2d})$; it hence holds $|s_s| = 4$.

These subscriptions contain the following predicates:

$$\begin{aligned}
\mathcal{P}(\mathcal{F}_{2a}) &= \{p_1, p_2, p_3, p_5, p_6\},\\
\mathcal{P}(\mathcal{F}_{2b}) &= \{p_1, p_2, p_3, p_7, p_8\},\\
\mathcal{P}(\mathcal{F}_{2c}) &= \{p_1, p_2, p_4, p_9, p_{10}\},\\
\mathcal{P}(\mathcal{F}_{2d}) &= \{p_1, p_2, p_4, p_{11}, p_{12}\}.
\end{aligned}$$

So let us count the number of conjunctive subscriptions per predicate. The number of conjunctions for predicates p_1 to p_{12} (in this order) is as follows: 4, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1, and 1. Building the average of these occurrences leads to $\frac{2\times4+2\times2+8\times1}{12} \approx 1.667$ and hence $|s_p| \approx 1.667$. Combining the former two canonical conversion parameters finally results in $s^{prop} = \frac{|s_p|}{|s_s|} = \frac{1.667}{4} \approx 0.417$.

Using our characterization framework, one should keep in mind that in practice conjunctive filtering algorithms might need to extend the created canonical form, which leads to higher values of $|s_p|$ (and thus s^{prop}). For example, the cluster algorithm can only cluster subscriptions that have the same numbers of predicates. Thus, it might need to insert "don't-care" predicates to allow for this property. We do not consider these increased memory requirements of conjunctive approaches in our framework but assume a more general setting (potentially leading to optimistic results for conjunctive algorithms).

5.2 Theoretical Analysis of the Counting Algorithm

Having presented our subscription characterization framework, we now model the memory requirements of the counting algorithm (see Section 2.3.3, page 40) with the help of this framework. In the two subsequent sections, we then analyze the cluster and the general Boolean approach.

In the following paragraphs, we analyze the memory requirements of both indexing and filtering structures. We start our individual observations with cases without predicate commonality ($p_c = 0$). Subsequently, we extend our analyses to more general settings involving common predicates.

Fulfilled Predicate Vector. The fulfilled predicate vector is required to store fulfilled predicates in the predicate matching step. In an implementation, one might apply an ordinary vector or a bit vector for this data structure. This decision should depend on the proportion of matching predicates. Let us consider a bit vector implementation in the following, requiring at least $\frac{|p|\times|s|}{8}$ bytes for no predicate commonality and $\frac{|p|\times|s|\times(1-p_c)}{8}$ bytes in general.

In cases of large numbers of fulfilled predicates per event, a bit vector implementation requires less memory than an ordinary vector implementation. However, if the proportion of fulfilled predicates per event and totally registered predicates is relatively small, utilizing an ordinary vector becomes advantageous.

Hit Vector. The hit vector accumulates the number of fulfilled predicates per subscription. For simplicity, let us assume a maximum number of 255 predicates per subscription[1]. Thus each entry in the hit vector requires 1 byte to represent the hit counter. Altogether, for $|s|$ registered subscriptions, which create $|s_s|$ conjunctions due to the canonical conversion, the space requirements are $|s| \times |s_s|$ bytes for the hit vector.

Because this vector consists of one entry per subscription, its memory usage is independent of predicate commonality p_c.

Subscription Predicate Count Vector. The counting algorithm stores the total number of predicates per subscription. According to our assumption for the hit vector, one can represent each subscription by a 1-byte entry in the subscription predicate count vector. Thus one needs $|s| \times |s_s|$ bytes in total due to the required canonical conversion.

Similarly to the hit vector, the subscription predicate count vector does not depend on predicate commonality (it consists of one entry per subscription).

Predicate-Subscription Association Table. An implementation of this table has to map each predicate to a list of subscriptions due to the required canonical conversion. This mapping to a list of subscriptions also holds in cases of no predicate commonality ($p_c = 0$). Least memory is demanded if predicate identifiers are used as indices in the predicate-subscription association table (this requires consecutive predicate identifiers). For maintaining the list of subscriptions, one has to store the corresponding number of subscription identifiers at a minimum, but we do not consider such implementation specifics

[1]This assumption can be easily relaxed.

here. Thus from an abstract viewpoint one only has to record the list of subscription identifiers (requiring $w(s) \times |s_p|$ bytes per predicate) for all registered predicates ($|p| \times |s|$ predicates in total), which demands $w(s) \times |s_p| \times |p| \times |s|$ bytes in total.

If considering predicate commonality p_c, for unique predicates (including one occurrence of each common predicate) the following amount of memory is required in bytes: $(1.0 - p_c) \times w(s) \times |s_p| \times |p| \times |s|$. Common predicates use $p_c \times w(s) \times |s_p| \times |p| \times |s|$ bytes. Thus predicate commonality does not influence the size of the predicate-subscription association table.

Subscription-Predicate Association Table. Least memory for subscription-predicate associations is required when using subscription identifiers as indices in the subscription-predicate association table. Each entry maps a subscription identifier to a list of predicate identifiers (there is also some implementation overhead as described for the predicate-subscription association table). Thus, one has to store a list of predicates for each subscription (there are $|s| \times |s_s|$ subscriptions in total due to conversion). Each list has to hold $|p| \times \frac{|s_p|}{|s_s|}$ predicate identifiers, which leads to $w(p) \times |s| \times |s_s| \times |p| \times \frac{|s_p|}{|s_s|} = w(p) \times |s| \times |p| \times |s_p|$ bytes in total for this table.

Predicate commonality p_c does not influence this data structure because it contains entries for each subscription. Thus common predicates do not allow for the storage of fewer associations between subscriptions and predicates.

Accumulating the previously determined memory usage leads to the following overall memory requirements:

$$\begin{aligned} mem_{counting} \quad = \quad & |s| \times (2|s_s| + w(s) \times |s_p| \times |p| + w(p) \times |s_p| \times |p| + \\ & \frac{|p| \times (1 - p_c)}{8}). \end{aligned} \tag{5.1}$$

5.3 Theoretical Analysis of the Cluster Algorithm

We now analyze a second conjunctive filtering algorithm with the help of our subscription characterization framework: the cluster algorithm [FJL+01]. As argued in Section 2.3.2 (page 37), this algorithm is a specialized filtering solution for applications that mainly involve highly common equality predicates.

Due to these strong application-dependent requirements, we cannot model the memory usage of all data structures of this algorithm with our framework, but instead focus on the most memory-consuming ones.

In our analysis, we again start by deriving the memory usage for cases without common predicates ($p_c = 0$). Subsequently, we extend our findings to the more general case, involving predicate commonality.

Predicate Bit Vector. This vector is similar to the fulfilled predicate vector applied in the counting algorithm. However, one always requires a bit vector implementation in this algorithm due to the need to access the state of predicates (whether they are fulfilled or not fulfilled) directly. Thus, one demands $\frac{|p|\times|s|\times(1-p_c)}{8}$ bytes for the predicate bit vector.

Cluster Vector. This vector contains references to subscription cluster lists. The number of entries depends highly on the actual number of access predicates. In turn, this number is dependent on the registered subscriptions and application semantics. Due to the unpredictability of such patterns, we neglect the memory requirements for this data structure in our following analysis. Furthermore, its memory usage is only a marginal proportion of the overall memory requirements.

Generally predicate commonality p_c results in a smaller cluster vector due to fewer access predicates (in fact, $p_c = 0$ contradicts the assumption of access predicates used in this algorithm).

Clusters. Subscriptions themselves are stored in clusters according to both their access predicates and their total number of predicates. Clusters consist of a subscription line storing an identifier for each subscription ($w(s)$ bytes required per subscription). Furthermore, they contain a predicate array holding the predicates each subscription consists of (on average requiring $\frac{|s_p|}{|s_s|}\times|p|\times w(p)$ bytes per subscription if only storing predicate identifiers). Clusters storing subscriptions with the same number of predicates and the same access predicates are linked together in a list structure. Here we neglect the memory requirements for this implementation-specific list.

Altogether clusters thus require $|s| \times |s_s| \times (w(s) + \frac{|s_p|}{|s_s|} \times |p| \times w(p))$ bytes to store $|s| \times |s_s|$ subscriptions. Predicate commonality does not influence the size of clusters. This results from the observation that clusters store predicates for all subscriptions. This storage happens in all cases of p_c and does not vary according to the commonality among predicates.

Subscription Cluster Table. This table is required to support efficient deregistrations. It allows for the determination of the cluster each subscription is stored in. When utilizing subscription identifiers as indices for the subscription cluster table, one requires $|s| \times |s_s| \times w(c)$ bytes for its storage of $|s| \times |s_s|$ cluster references.

This table is also focused on mappings from subscriptions. Thus its size is independent of predicate commonality p_c.

Predicate-Subscription Association Table. An association between predicates and subscriptions is required to allow for an efficient support of deregistrations. If using predicate identifiers as indices in the table (or storing associations inside indexes), one requires $w(s) \times |s_p| \times |p| \times |s|$ bytes for these associations of $|p| \times |s|$ predicates, each being contained in $w(s) \times |s_p|$ subscriptions on average.

Similar to our observation for the counting algorithm, for unique predicates one requires $(1.0 - p_c) \times w(s) \times |s_p| \times |p| \times |s|$ bytes to store their associations with subscriptions. Common predicates consume $p_c \times w(s) \times |s_p| \times |p| \times |s|$ bytes. Therefore predicate redundancy does not influence the size of predicate-subscription associations in this table.

Accumulating the previously determined memory usages leads to the following overall memory requirements:

$$\begin{aligned} mem_{cluster} &= |s| \times (|s_s| \times (w(c) + w(s)) + |s_p| \times |p| \times (w(s) + w(p)) + \\ &\quad \frac{|p| \times (1 - p_c)}{8}). \end{aligned} \tag{5.2}$$

5.4 Theoretical Analysis of the General Boolean Algorithm

The final algorithm we analyze is our general Boolean filtering approach (see Chapter 4), obviously not requiring canonical conversion. Following our previous procedure, we start by analyzing the memory requirements without common predicates and then extend our findings to the general case, involving shared predicates ($p_c > 0$). Because the general Boolean approach extends the counting algorithm, some of the required data structures are identical to the counting approach. However, their memory requirements differ because no canonical conversion needs to be applied.

Fulfilled Predicate Vector. This vector serves the same purpose as its counterpart in the counting algorithm. It can be realized as an ordinary vector or as a bit vector. A bit vector implementation requires $\frac{|p|\times|s|\times(1-p_c)}{8}$ bytes of memory.

Subscription Trees. The encoding scheme of subscription trees was presented in Section 4.2.2 (page 102). For predicates, stored in leaf nodes, one requires $|p| \times (w(p) + 1)$ bytes per subscription. Inner nodes, containing the Boolean operators and the numbers of children, demand $2|op|$ bytes of memory for each subscription. Thus, for all registered subscriptions $|s| \times ((w(p) + 1) \times |p| + 2|op|)$ bytes are required.

Subscription trees have to store operators and predicate identifiers in all cases. Thus, they do not depend on the commonality among predicates p_c.

Subscription Location Table. This table is applied to associate subscription identifiers and subscription trees. If utilizing subscription identifiers as indices in this table, ones requires $w(l) \times |s|$ bytes for storing these associations.

Because the subscription location table contains entries per subscription, its memory usage is not influenced by predicate commonality p_c.

Predicate-Subscription Association Table. The predicate-subscription association table requires less memory than its counterparts in the two previously analyzed algorithms. This is implied by the fact that subscriptions do not need to be converted to canonical forms by the Boolean approach. Thus predicates are involved in fewer subscriptions (e.g., always only one subscription in case of p_c=0). Altogether, $|s| \times |p| \times w(s)$ bytes are required for the predicate-subscription association table.

In cases of predicate commonality, one requires $p_c \times |s| \times |p| \times w(s)$ bytes for unique predicates. Moreover, redundant predicates consume $(1.0 - p_c) \times |s| \times |p| \times w(s)$ bytes. In summary, the memory usage of the predicate-subscription association table does not depend on the commonality among predicates.

Hit Vector. According to the hit vector in the counting approach, this data structure is used to accumulate the number of fulfilled predicates per subscription. Because no conversion to canonical expressions is required by the Boolean algorithm and according to the common assumption of a maximum of 255 predicates per subscription, the hit vector requires $|s|$ bytes of memory.

The memory requirements do not depend on predicate commonality because the hit vector contains entries per subscription.

Minimum Predicate Count Vector. According to our assumption of a maximum of 255 predicates per subscription, the minimum predicate count vector requires $|s|$ bytes of memory.

Equivalently to the hit vector, this data structure does not depend on the commonality among predicates.

Summing up the identified memory requirements for the individual data structures, we derive the following overall memory usage:

$$\begin{aligned} mem_{Boolean} &= |s| \times (|p| \times (1 + w(p) + w(s)) + 2|op| + w(l) + 2 + \\ &\quad \frac{|p| \times (1 - p_c)}{8}). \end{aligned} \tag{5.3}$$

5.5 Theoretical Algorithm Comparison

Having described the memory requirements of two conjunctive filtering algorithms and our general Boolean representative, we now compare the memory usage of the conjunctive solutions to the general Boolean one. When only considering the filtering component, these memory requirements directly affect the scalability of a solution (see Section 2.2 and Section 2.6 on page 23 and page 58, respectively). From our following analysis, we can thus also deduce under what circumstances a Boolean filtering algorithm should be preferred with respect to scalability and the settings that favor a conjunctive solution.

5.5.1 Point of Interchanging Memory Requirements

In the subsequent comparison, we directly use the accumulated memory requirements of the three algorithms, given in Equations 5.1 to 5.3 (page 128, 130, and 132). The memory usage in all three cases grows linearly with the number of subscriptions $|s|$ (and is zero if no subscriptions are registered).

Hence, we only need to analyze the first derivatives of the functions in Equations 5.1 to 5.3 at $|s|$ for a comparison of the memory requirements. For the counting algorithm (Equation 5.1), it thus holds:

$$\begin{aligned} mem'_{counting}(|s|) &= 2|s_s| + w(s) \times |s_p| \times |p| + w(p) \times |s_p| \times |p| + \\ &\quad \frac{|p| \times (1 - p_c)}{8}. \end{aligned} \tag{5.4}$$

Similarly for the cluster algorithm (Equation 5.2) we derive:

$$\begin{aligned} mem'_{cluster}(|s|) &= |s_s| \times (w(c) + w(s)) + |s_p| \times |p| \times (w(s) + w(p)) + \\ &\quad \frac{|p| \times (1 - p_c)}{8}. \end{aligned} \tag{5.5}$$

Finally, the general Boolean approach (Equation 5.3) leads to the following first derivation:

$$\begin{aligned} mem'_{Boolean}(|s|) &= |p| \times (1 + w(p) + w(s)) + 2|op| + w(l) + 2 + \\ &\quad \frac{|p| \times (1 - p_c)}{8}. \end{aligned} \tag{5.6}$$

To reduce the number of variables in these equations, let us now assume typical values for the algorithm-specific parameters, as stated in Section 5.1.3: $w(s) = 4$, $w(p) = 4$, $w(l) = 4$, and $w(c) = 4$. That is, the widths of subscription identifiers, predicate identifiers, subscription locations, and cluster references are 4 bytes each[2]. Finally, let us further reduce the number of variables by utilizing the proportional notions of op^{prop} (proportional number of operators) and s^{prop} (proportional number of conjunctive elements per predicate), as defined in Section 5.1.

With these specifications, we now compare the memory usage of the conjunctive algorithms (Equation 5.4 and Equation 5.5) to that of the general Boolean approach (Equation 5.6). The following inequalities denote the points where the general Boolean approach requires less memory for its event filtering data structures than the respective conjunctive solution. These points are described in terms of the characterizing parameter $|s_s|$. That is, the general Boolean approach requires less memory if a canonical conversion to disjunctive normal form creates more than the stated number of conjunctive subscriptions.

[2]These values hold on 32-bit machines when using standard (unsigned) integers as identifiers and standard memory pointers.

We refer to these points as *turning points* because they describe in what cases of $|s_s|$ a general Boolean filtering algorithm becomes worthwhile. To allow for a better overview, we use the notation $|s_s|(\frac{algorithm}{Boolean})$ to denote the conjunctive algorithm "algorithm" compared to the general Boolean approach:

$$|s_s|(\frac{counting}{Boolean}) \quad > \quad \frac{|p| \times (2op^{prop} + 9) + 6}{2 + 8s^{prop} \times |p|}, \tag{5.7}$$

$$|s_s|(\frac{cluster}{Boolean}) \quad > \quad \frac{|p| \times (2op^{prop} + 9) + 6}{8 + 8s^{prop} \times |p|}. \tag{5.8}$$

Having found these turning points, we illustrate them graphically in the following subsection.

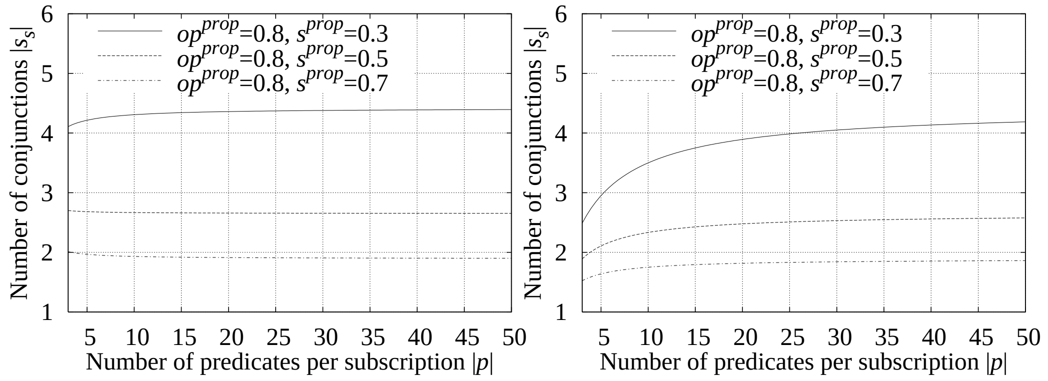
5.5.2 Graphic Illustration of the Turning Point

Figure 5.1 shows the turning point for different parameter combinations. The turning point when comparing the counting and the general Boolean approach is illustrated in Figure 5.1(a); Figure 5.1(b) depicts the cluster in comparison to the general Boolean algorithm. On the abscissae of the figures, we show the number of predicates per subscription $|p|$. The ordinates show the number of conjunctions $|s_s|$ that need to be created by canonical conversion to lead to a more space-efficient general Boolean filtering approach.

In the figures, we vary s^{prop} (proportional number of conjunctions per predicate) between 0.3 and 0.7 to show the influence of this parameter on the turning point. Our example subscription classes show values of s^{prop} between approximately 0.4 and 0.7 (see Section 5.1.4). For parameter op^{prop} (proportional number of operators), we choose $op^{prop} = 0.8$ in these figures, being approximately the average of op^{prop} in our example classes (in between approximately 0.7 and 0.9).

To interpret Figure 5.1, one chooses one conjunctive algorithm (i.e., either Figure 5.1(a) or Figure 5.1(b)), one of the curves (specifying op^{prop} and s^{prop}), and the number of predicates $|p|$ on the abscissa. One then gets the mapping for this scenario on the ordinate, specifying the turning point. We demonstrate this process in the following example:

Example 5.2 (Finding the turning point) *To determine the turning point for the counting algorithm in comparison to the general Boolean approach, we*



(a) Counting and Boolean (b) Cluster and Boolean

Figure 5.1: Turning point (point of interchanging memory requirements) for counting and Boolean approach, and cluster and Boolean approach.

need to consider Figure 5.1(a). Let us assume that subscriptions, on average, specify 10 predicates ($|p| = 10$*). We thus fix the value "10" on the abscissa. Let us further assume that the number of operators proportional to the number of predicates is approximately 0.8 (*$op^{prop} = 0.8$*), and that after the conversion a predicate occurs in approximately 70 percent of the created conjunctions (*$s^{prop} = 0.7$*).*

*Thus we find the turning point as the value of the lowermost curve in Figure 5.1(a) for argument *$|p| = 10$*: the number of conjunctions that is created by the conversion has to be less than two (*$|s_s| < 2$*). Hence whenever a subscription is not purely conjunctive, the general Boolean algorithm requires less memory than the counting approach (and is thus more scalable) for this scenario.*

From the viewpoint of the general Boolean approach, the lower a curve is situated in Figure 5.1, the more advantageous this algorithm performs in comparison to a conjunctive solution. The reason for this property is that already an only slightly increased complexity after a canonical conversion leads to less memory use for the Boolean approach. From the viewpoint of conjunctive algorithms, conversely, the higher a curve is located in comparison to other conjunctive approaches, the more space-efficient is the respective solution.

For both conjunctive algorithms, an increase in the number of operators (increasing op^{prop}) when holding the other parameters fixed leads to more space efficiency compared to the Boolean approach. This is because the Boolean algorithm needs to encode and store these operators, but not the conjunctive solutions. When observing the other parameter, the number of conjunctions per predicate after conversion s^{prop}, an increase in this parameter leads to a more space-efficient Boolean approach. Obviously this is founded in the fact that the complexity of the converted conjunctive subscriptions increases if the other parameters stay fixed.

Comparing the counting and cluster approach, the counting algorithm is more space-efficient than the cluster algorithm (curves for the same parameter setting are located higher in Figure 5.1(a) than in Figure 5.1(b)). In particular, for small predicates numbers (left on the abscissa), the counting approach outperforms the cluster algorithm. The reason for this behavior is the requirement to store subscription cluster table and subscription identifiers in clusters regardless of the number of predicates, which leads to a larger proportional memory use for an overall small number of predicates $|p|$. For higher predicate numbers, both conjunctive algorithms lead to comparable turning points. The counting algorithm, though, always stays slightly more space-efficient than the cluster algorithm.

5.5.3 Properties of Example Subscription Classes

In this dissertation, we focus on general-purpose filtering algorithms for pub-sub systems. After our conceptual analysis of the turning point, we now compare the memory requirements of our example subscription classes using the counting and our general Boolean approach.

To determine the preferable general-purpose filtering algorithm for subscriptions of these classes, we use the findings from Section 5.1.4, describing these classes with the help of our subscription characterization framework. Having determined the turning point with the help of Equation 5.7, we can compare this point to the real number of conjunctions $|s_s|$ that is created by the canonical conversion of these classes. If $|s_s|$ is greater than or equal to the derived turning point, the general Boolean filtering solution is favorable with respect to memory requirements. For Subscription Class 1, we derive the following turning point:

$$|s_s|(\frac{counting}{Boolean}) \quad > \quad \frac{6 \times (2 \times 0.667 + 9) + 6}{2 + 8 \times 0.667 \times 6} = \frac{68}{34} = 2.$$

The turning point for Subscription Class 2 is as follows:

$$|s_s|(\frac{counting}{Boolean}) \quad > \quad \frac{12 \times (2 \times 0.833 + 9) + 6}{2 + 8 \times 0.417 \times 12} = \frac{134}{42} \approx 3.19.$$

Finally, for Subscription Class 3 the turning point is:

$$|s_s|(\frac{counting}{Boolean}) \quad > \quad \frac{7 \times (2 \times 0.857 + 9) + 6}{2 + 8 \times 0.429 \times 7} = \frac{81}{26} \approx 3.12.$$

The actual created number of conjunctive subscriptions for these classes is given in Table 5.2 (page 125): $|s_s| = 2$ for Class 1, $|s_s| = 4$ for Class 2, and $|s_s| = 6$ for Class 3. Hence for all subscription classes the general Boolean algorithm requires less (Class 2 and 3) or equal (Class 1) memory than the conjunctive solution. Consequently, for our online auction scenario one should apply a Boolean filtering approach with respect to memory use.

5.6 Practical Algorithm Comparison

Having the theoretical means to determine whether a conjunctive or a Boolean filtering algorithm is preferable for a given setting, we now verify our findings by experiment. In this empirical evaluation, we compare the counting algorithm and our general Boolean approach, in accordance with the focus of this dissertation.

5.6.1 Experimental Setup

A practical implementation of filtering algorithms requires memory resources additional to those described by our theoretical framework. For example, one needs suitable data structures that efficiently support the required operations and these structures need extra memory for their effective management. Thus, in a practical implementation, one has to face a higher memory cost than described in the theoretical model.

Also, the data structures have to be implemented reasonably. For example, indexing structures (e.g., predicate-subscription association table) require a dynamic implementation to allow for both registrations and deregistrations. For sole filtering structures (e.g., fulfilled predicate vector), on the other hand, it is sufficient to provide static implementations.

The experimental testbed we use to confirm our theoretical findings only contains implementations of the subscription indexing parts of both counting and general Boolean algorithm (both candidate and final subscription matching for the Boolean approach). We exclude predicate indexes because both approaches can apply the same indexes for this purpose and thus require the same memory in practice.

For the dynamic data structures of the algorithms, we used dynamic array implementations that consumed less memory than their STL[3] variants in empirical studies. These dynamic structures include the required tables (e.g., predicate-subscription association table), whose implementation is also based on our dynamic array.

Because the general Boolean approach extends the counting algorithm and because of our choice to use comparable implementations for both of these approaches, our experiments reveal whether the practical memory overhead is comparable for these two classes of filtering algorithms. That is, we can verify the findings of our theoretical framework with the provided practical implementation.

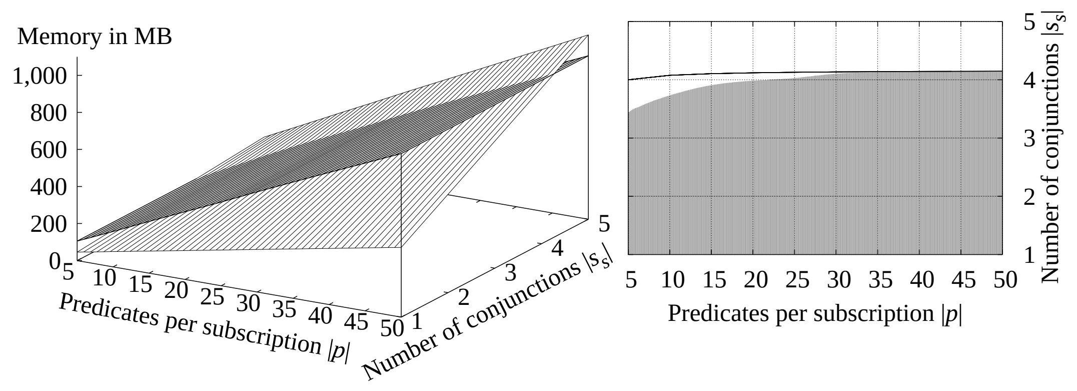
Our experiments required us to use an artificial test setup to derive data points for a wide range of parameter assignments. We analyzed predicate numbers in the interval from $|p| = 5$ to $|p| = 50$. The number of conjunctive subscriptions due to conversion was varied between $|s_s| = 1$ and $|s_s| = 5$. We also used different numbers of operators $|op|$ and conjunctions per predicate $|s_p|$ in our experiments. Here we present the results for the setting $|op| = 0.5$, and the three assignments $|s_p| = 0.3$, $|s_p| = 0.5$, and $|s_p| = 0.7$. The turning point is generally independent of the number of subscriptions; in our experiments we used 1,000,000 subscriptions ($|s| = 1,000,000$).

In the following, we report the total memory requirements of the filtering process using information provided by the process status application programming interface (PSAPI).

5.6.2 Illustrating the Memory Usage

Figure 5.2(a) illustrates the memory requirements (z-axis) for the counting algorithm (light surface) and the general Boolean algorithm (dark surface). The surface that represents the counting algorithm is derived (i.e., interpolated) from 50 points (10 values for $|p|$ on the x-axis and five values for $|s_s|$ on the y-axis). The surface that illustrates the general Boolean approach is derived

[3]Standard Template Library [SL95].



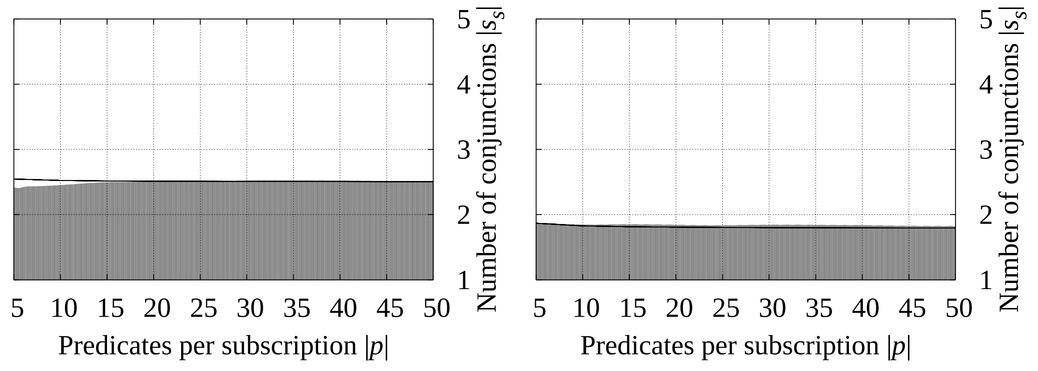
(a) Perspective view (b) Top view

Figure 5.2: Memory requirements for counting algorithm (light surface) and Boolean algorithm (dark surface) for the setting $|s| = 1,000,000$, $op^{prop} = 0.5$, and $|s_p| = 0.3$. In the right figure, we show the same setting and a top view to the left figure. The light surface is illustrated transparently in the right figure and our theoretical result is indicated by the additional curve.

from the 10 different values for $|p|$ (x-axis). The memory requirements of this algorithm are independent of conversion and thus have the same values for all assignments of $|s_s|$ (y-axis).

As illustrated in Figure 5.2(a), the specialized counting algorithm requires less memory than the Boolean approach for small values of $|s_s|$. However, the more conjunctions are created due to conversion (higher values on y-axis), the higher the memory usage of the counting approach (z-axis). One can directly observe that both surfaces cut at some point: the turning point, theoretically described in Equation 5.7.

To get a better overview of the turning point, we illustrate a top view of the behavior of the algorithms in the described setting in Figure 5.2(b). We remove the surface that represents the counting algorithm and just show the surface for the Boolean approach in this figure. This surface is shown until it is cut by the surface of the counting algorithm and thus covered in the illustrated top view. In Figure 5.2(b), we additionally illustrate a curve that represents the theoretically derived turning point in the same setting. For our two other settings, we show a similar top view to the empirical results in Figure 5.3(a) ($|s_p| = 0.5$) and Figure 5.3(b) ($|s_p| = 0.7$). There we also included the theoretically determined turning point.

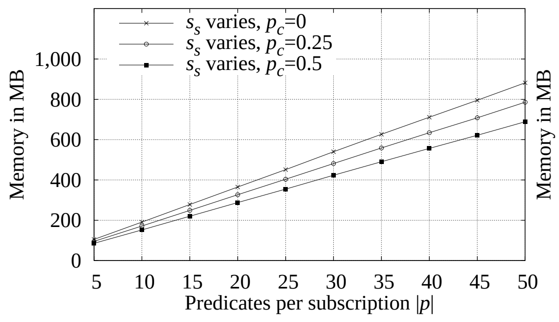


(a) Top view, $|s_p| = 0.5$ (b) Top view, $|s_p| = 0.7$

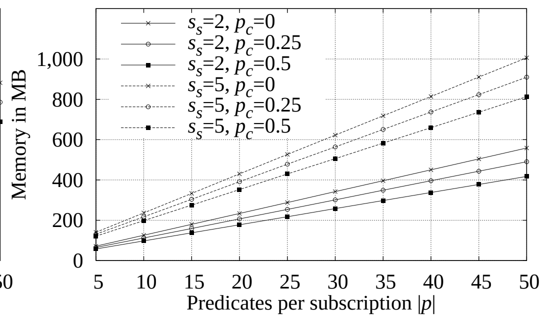
Figure 5.3: Turning point for the setting $|s| = 1,000,000$, $op^{prop} = 0.5$, and varying values of $|s_p|$. The theoretical result is indicated by the additional curve.

The theoretically predicted turning point broadly aligns with the behavior in practice in all three of these figures. However, for small predicate numbers (left on the abscissae) the turning point in practice can be found below the theoretically determined one. This is particularly the case for small values of $|s_p|$ (cf. Figure 5.2(b)). This behavior, in fact, means that for small predicate numbers the general Boolean approach leads to even better results in practice than in theory: even disjunctive normal forms less complex than predicted by the theoretical characterization framework do already favor a general Boolean filtering solution.

The reason for this behavior is found in the data structures for these algorithms: the subscription-predicate association table that is required in the counting algorithm has a relatively high management overhead for small predicate numbers and small values of $|s_p|$ because the created conjunctive subscriptions involve an even smaller number of predicates in this case. Hence the memory use for management purposes proportional to the data in this table is relatively high. However, this is not the case for the subscription trees in the Boolean approach. For larger predicate numbers in the created conjunctions, the proportion of memory for management and stored data gets smaller, and becomes comparable in both approaches.



(a) General Boolean algorithm



(b) Counting algorithm

Figure 5.4: Influence of predicate commonality on the general Boolean algorithm and the counting algorithm, using the setting $|s| = 1,000,000$, $op^{prop} = 0.5$, and $s^{prop} = 0.3$.

5.6.3 Predicate Commonality

Although our theoretical analysis shows that the memory requirements do only marginally depend on predicate commonality p_c (only the fulfilled predicate vector is influenced by p_c), the behavior in practice is different. The reason for this development is again found in the varying overhead for the management of data structures in an implementation, in this case for the predicate-subscription association table that is required in both algorithms. The result is a decreasing memory usage for increasing predicate commonality p_c.

Figure 5.4(a) shows this behavior for the Boolean algorithm; the counting approach is illustrated in Figure 5.4(b). The abscissae of these figures state the number of predicates $|p|$. The memory usage is displayed at the ordinates. In this set of experiments, we use the following parameters: $|s| = 1,000,000$, $op^{prop} = 0.5$, and $s^{prop} = 0.3$. The curves in the figures state different predicate commonalities: $p_c = 0$, $p_c = 0.25$, and $p_c = 0.5$. For the counting algorithm (Figure 5.4(b)), we illustrate two settings, $|s_s| = 2$ and $|s_s| = 5$.

Although predicate commonality p_c changes the memory use for both algorithms in practice, this effect does not influence the turning point of the memory requirements (see Section 5.5.2): predicate commonality has the same effect on the Boolean and the conjunctive algorithm, as shown in Figure 5.4.

5.7 Correlation to Filter Efficiency

We now investigate the time efficiency properties of both filtering solutions. Combining our findings regarding time and memory efficiency allows for a full overview of the advantages and disadvantages of these categories of algorithms.

5.7.1 Experimental Setup

The efficiency properties of filtering algorithms depend on the settings that are chosen for empirical studies. For this set of experiments, we decided to comparatively analyze the counting and the general Boolean algorithm in conjunction with subscriptions of the three subscription classes that were identified in Section 3.3 (page 79). Additionally, event messages follow our findings from Chapter 3. Hence, we apply the derived semi-realistic dataset for our experiments.

Regarding the variable parameters for the creation of event messages (see Section 3.2.2, page 76), we evaluated various assignments for these variables. Here, we present the results using $B_{prop} = 100$, $A_{prop} = 5$, $p^{A}_{mult} = 0.1$, and $p^{T}_{mult} = 0.01$. The results presented here were derived by publishing $100,000$ event messages, leading to stable averages. Event messages were created at the beginning of each experiment and published after the previous message was processed.

For the distributions of the operands of variable predicates of subscriptions, we assume five different settings: uniform distribution, normal distribution (minimum value has the highest probability), Zipf distribution (minimum value has the highest probability), reversed normal distribution (maximum value has the highest probability), and reversed Zipf distribution (maximum value has the highest probability). For the exact ranges of the operands of these predicates and details about predicate distributions, we refer to Appendix B.2; Appendix B.1 describes the mapping of attribute domains to data types in our experiments.

We used our pub-sub prototype, BoP, in this set of experiments. The central broker is run on a machine equipped with 512 MB of RAM and a 2 GHz processor. The BoP prototype is implemented in C/C++. Predicate indexes are realized using the Stl `map` class. Minimum predicate count vector and subscription location table are based on the Stl class `vector`. For predicate-subscription association table, and fulfilled predicate and hit vector, we used dynamic array implementations (see Section 5.6.1). In this centralized version of BoP, one cannot apply most of the algorithm optimizations that were presented in Section 4.5. For this set of experiments, BoP only uses the short-circuiting method (see Section 4.5.2, page 112). For the evaluation of the other optimizations, we refer to our experiments in the distributed setting in Chapter 8.

As argued before (Section 4.3, page 107), the filtering algorithm in BoP supports conjunctive subscriptions in nearly the same way as the counting approach. We compiled a conjunctive version of BoP for our comparative evaluation, removing the overhead of accessing subscription location table and subscription trees. Proceeding in this way ensures that both algorithms utilize the same data structures and thus removes implementation-specific influences. The conjunctive version of BoP performs the canonical conversion before registering subscriptions and then applies the original counting approach.

5.7.2 Filtering of Example Subscription Classes

Figure 5.5 shows the filter efficiency (ordinate) of both algorithms ("Bool" and "Conj" in the figure) with an increasing number of registered subscriptions $|s|$ on the abscissa. For each algorithm, we evaluated five distributions in the operands of subscriptions. As can be seen in the figure, the time efficiency of both algorithms is only slightly influenced by the actual predicate distributions.

Theoretically, both algorithms should show linearly increasing filter times (ordinate) with increasing subscription numbers (abscissa). In practice, however, both approaches appear to lead to super-linearly developing filter times, as illustrated in Figure 5.5. The reason for this property of both algorithms is found in their general approach of incrementing counters per subscription and the influence of a limited processor cache. The behavior of the filter efficiency, in fact, is linear but advantageously influenced by the processor cache for small subscription numbers, leading to a smaller initial gradient.

The point of changing gradients occurs at a much smaller number of subscriptions for the counting algorithm than for the general Boolean algorithm because the conjunctive algorithm needs to internally convert the original subscriptions. Thus, for the counting algorithm, more counters in more subscriptions need to be increased after the conversion (i.e., the hit vector contains more entries). Hence the hit vector does not fit into the processor cache from approximately 50,000 original subscriptions onwards (leading to 200,000 converted ones). After having registered approximately 100,000 original subscriptions (400,000 converted ones), the influence of the processor cache is negligible and thus the maximal gradients of the curves are reached.

Although the general Boolean algorithm is subjected to the same influence,

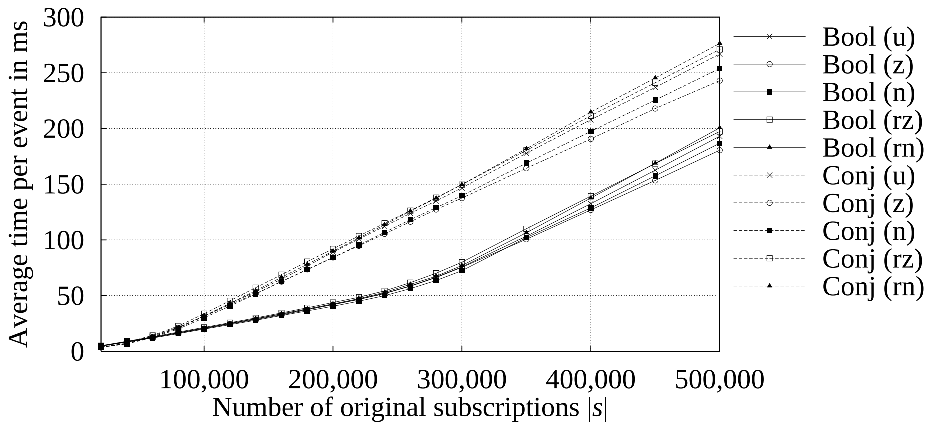


Figure 5.5: Filter efficiency of the Boolean algorithm ("Bool") and the conjunctive counting algorithm ("Conj") in the combined setting using various distributions in predicates (u–uniform, n–normal, z–Zipf, rn–reversed normal, rz–reversed Zipf distribution).

the effect on this algorithm is much less:

Firstly, the point of changing gradients occurs at a much larger number of registered subscriptions: $|s| \approx 200,000$ on the abscissa. Interestingly, this number of subscriptions is four times the number of subscriptions in the conjunctive setting. The reason for this behavior is clearly that conversion leads to four times the number of original subscriptions. Hence the processor cache can store four times more unconverted subscriptions in the hit vector.

Secondly, the main proportion of filter time is not spent on increasing counters but on evaluating candidate subscriptions (there are far fewer predicates because no canonical conversion needs to be performed). Hence the processor cache only has a minor influence on overall filter efficiency, that is, the change in the gradient is less than in the conjunctive algorithm.

Interpreting the curves in Figure 5.5, both algorithms initially show similar filter times. The more subscriptions get registered, the larger the difference between general Boolean approach and conjunctive approach. Having registered more than 400,000 subscriptions, the difference between the algorithms stabilizes: per event message, the Boolean solution requires approximately 75, 64, 68, 72, and 76 milliseconds less than the conjunctive approach for the five tested distributions. For 500,000 subscriptions, this is an improvement of approximately 27 percent for the analyzed predicate distributions.

Although the presented results depend on the processor cache, they can be generalized to universal settings: Changes in the cache size only shift the

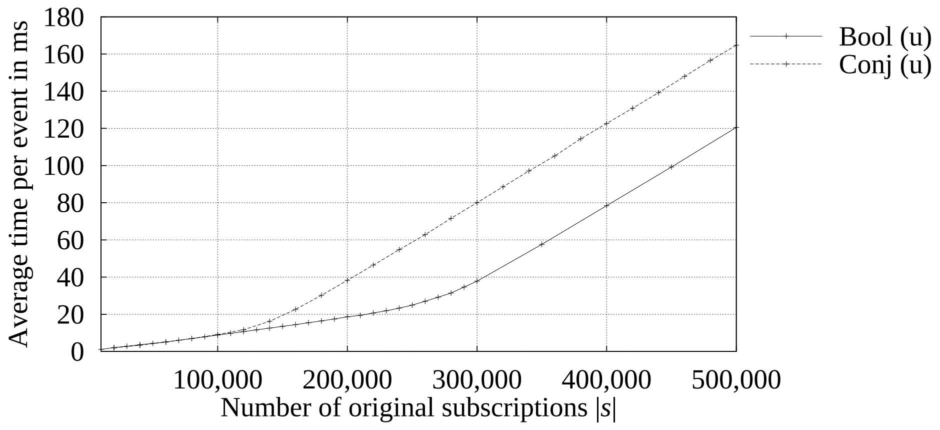


Figure 5.6: Filter efficiency for Subscription Class 1 using uniform distributions in predicates.

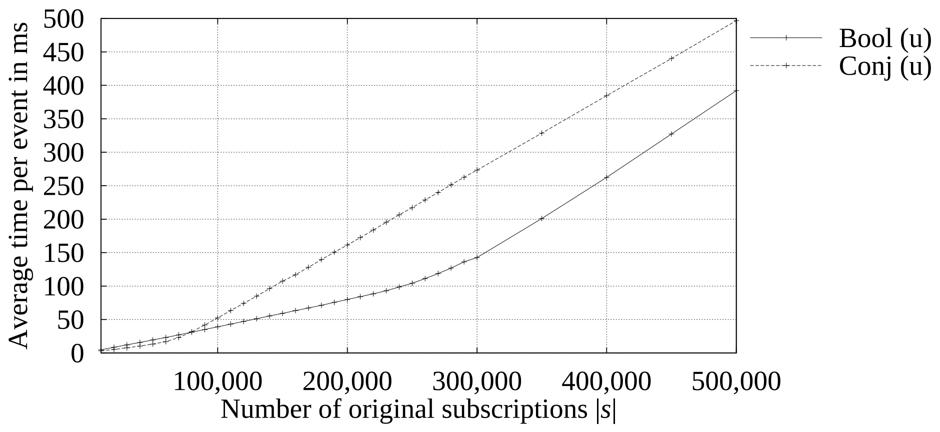


Figure 5.7: Filter efficiency for Subscription Class 2 using uniform distributions in predicates.

point of changing gradients (on the abscissa) for both filtering approaches. The occurrence of this point for the conjunctive algorithm always happens at a far smaller number of subscriptions due to the need to perform canonical conversion for such an approach. For large subscription numbers, the influence of the processor cache on filter time is negligible, as can be seen in the increased but stable gradients of the curves. Here the general Boolean filtering approach is more time-efficient than the conjunctive approach, as shown in Figure 5.5.

Separate Settings: Individual Subscription Classes

The time efficiency of both counting algorithm and general Boolean algorithm depends on the registered subscriptions. Having evaluated the combined setting in the previous paragraph, we now investigate the filter efficiency for the

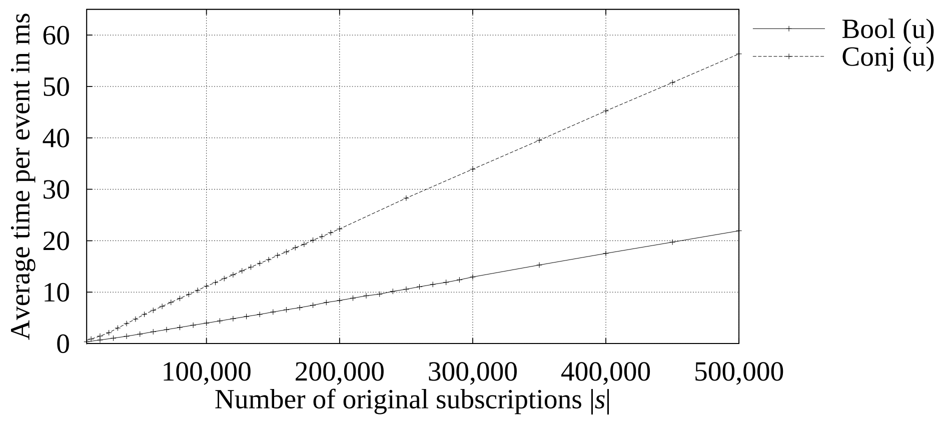


Figure 5.8: Filter efficiency for Subscription Class 3 using uniform distributions in predicates.

individual subscription classes.

Figures 5.6 to 5.8 give an overview of the time efficiency properties of both algorithms. For all subscription classes, one can identify the points of changing gradients. The difference in these points broadly aligns with the influence of canonical conversion that is required for the counting approach: two, four, and six conjunctive subscriptions are created due to conversion. Hence, in the Boolean setting, the point of changing gradients occurs at a number of registered subscriptions that is approximately two, four, and six times higher than in the conjunctive setting.

Subscription Class 1 (Figure 5.6) and Class 3 (Figure 5.8) always show similar or advantageous efficiency properties in the general Boolean approach. In particular for Subscription Class 3, the difference between the two algorithm classes (Boolean and conjunctive) constantly increases for a growing subscription base: for 500,000 registered subscriptions, the general Boolean approach is approximately 61 percent more time-efficient than the conjunctive approach.

For Subscription Class 1, the time difference between the approaches remains nearly constant from approximately 300,000 subscriptions onwards (the Boolean solution shows a slightly smaller gradient). For subscriptions of this class, the general Boolean approach is approximately 27 percent more efficient than the conjunctive approach for 500,000 registered subscriptions.

For Subscription Class 2 (Figure 5.7) up to approximately 80,000 subscriptions, the counting approach is slightly more efficient than its general Boolean extension. However, this behavior changes for higher subscription numbers, where the Boolean algorithm becomes the more efficient solution. The final gradients of both algorithm are nearly on par with each other (the conjunctive solution shows a slightly smaller final gradient[4]). For 500,000 registered subscriptions, the general Boolean filtering solution is approximately 20 percent more efficient than the conjunctive approach.

Theoretically, the difference between conjunctive and Boolean algorithm should increase with an increasing number of subscriptions created due to conversion (i.e., with Subscription Class 1, 2, and 3). Subscription Class 2, however, does not follow this trend. The reason is that for this class of subscriptions the general Boolean filtering algorithm leads to a large number of candidate subscriptions to evaluate. Despite this effect, the Boolean algorithm still leads to a higher filter efficiency than its conjunctive counterpart.

The differing behavior of filter efficiency for subscriptions of the individual classes leads to the results in the combined setting. Here, the Boolean algorithm is approximately 27 percent more efficient than the conjunctive algorithms, as presented in the previous paragraph.

5.8 Summary

In this chapter, we presented a comparative evaluation of the general-purpose conjunctive counting algorithm and our general-purpose Boolean filtering algorithm. This analysis focused on the two quality measures system efficiency and scalability that were identified in Section 2.2. We introduced a characterization framework to describe the typical patterns of subscriptions. Based on this framework, we then described the memory use of two conjunctive algorithms and our general Boolean algorithm. We found that for our example subscription classes a general Boolean algorithm requires less memory than conjunctive approaches. Our solution is thus the favorable choice with respect to scalability for an online auction scenario.

Generally we could show that there are various settings in which the occurrence of only one disjunction in subscriptions favors a Boolean filtering solution. Thus, general Boolean filtering algorithms are the more memory-aware and therefore the more scalable choice for the central filtering components if subscriptions are not purely conjunctive. Our characterization framework allows us to choose the preferable algorithm class for any subscriptions.

[4]However, the conjunctive solution does not scale to the number of subscriptions that the Boolean solution does (see Section 5.5.3).

Regarding the comparison of time efficiency, we analyzed the behavior in the online book auction example scenario. We evaluated several predicate distributions in subscriptions and found that the general Boolean filtering algorithm is also favorable over a general-purpose conjunctive algorithm with respect to filter efficiency for large subscription numbers. We can thus state that a general Boolean solution fulfills our design goals to be the preferable algorithm class for subscriptions that are more general than conjunctive subscriptions in universal settings: it is firstly more space-efficient, and secondly equally or more time-efficient. We therefore proved the first part of our central hypothesis (page 6):

> *In general-purpose pub-sub systems, a general Boolean filtering approach requires less memory and achieves higher filter efficiency than a conjunctive filtering approach.*

Having the means to support the central filtering of general Boolean subscriptions, we also need to support this class of subscriptions in the routing algorithms of content-based pub-sub systems. We proceed with this step in the next chapter.

Chapter 6

Routing Optimizations for General Boolean Subscriptions

The support of general Boolean subscriptions in a filtering algorithm, as proposed in Chapter 4 and evaluated in Chapter 5, is only the first step towards the provision of a content-based pub-sub system for a general Boolean pub-sub model. The second step concerns the routing in the distributed system, in particular the subscription-based routing optimizations that currently only support conjunctive subscriptions and thus are not applicable (see Section 2.5, page 47).

We take this step within this chapter and propose the first subscription-based routing optimizations for general Boolean subscriptions, including restricted conjunctive forms. The proposed optimizations follow a different optimization principle than current solutions, allowing for the combination of our novel and recent optimizations. For the following descriptions, we assume subscription forwarding or rendezvous nodes (see Section 2.4, page 42) as routing algorithms.

The structure of this chapter is as follows: in Section 6.1 we introduce the general idea and principle of our optimization proposals. The first optimization, predicate replacement, is presented in Section 6.2. Subscription pruning, our second and more advanced optimization, is proposed in Section 6.3. Our subscription pruning approach allows for the optimization of content-based pub-sub systems with respect to various target parameters. In Section 6.4 we describe how to tailor the general subscription pruning optimization to these parameters. Subscription pruning can be applied in three different ways; they are investigated in Section 6.5. The practical implementation of the optimization is the focus of Section 6.6. Finally, we present related work in Section 6.7.

6.1 Optimization Idea

Current subscription-based routing optimizations, that is, subscription covering, merging, and summarization, aim at reducing the number of event routing table entries, as described in Section 2.5. However, there are several disadvantages of these approaches. Firstly, the potential of such a reduction depends on the existence of certain relationships (e.g., subset relationships or similarities) among the registered subscriptions. Secondly, the discovery of these relationships mostly requires the relation of all subscriptions with each other, leading to complex computation problems for general Boolean expressions. Thirdly, having applied these recent optimizations, the deregistration of subscriptions might require a network- and time-consuming processing. We refer to Section 2.5 for a detailed overview of these optimizations and an analysis of their properties.

In this chapter we follow a different optimization idea than current approaches. Our design goals are to solve the common problems of existing routing optimizations, as summarized before and identified in Section 2.5.5 (page 58). We want to provide an optimization that:

1. is applicable to all kinds of Boolean subscriptions, including restricted conjunctive ones.
2. does not depend on the covering relationships among subscriptions.
3. does not increase the complexity of deregistrations compared to unoptimized routing.
4. increases the overall routing efficiency of the system.
5. decreases the memory requirements for event routing tables.

Fulfilling these goals, we can ultimately prove Part 2a of our central hypothesis (page 6):

> *Subscription pruning increases system efficiency and decreases routing table size, independently of the existing covering relationships.*

To achieve our design goals, our optimizations manipulate the entries in event routing tables on an individual basis instead of relating them to each other, as current optimizations do.

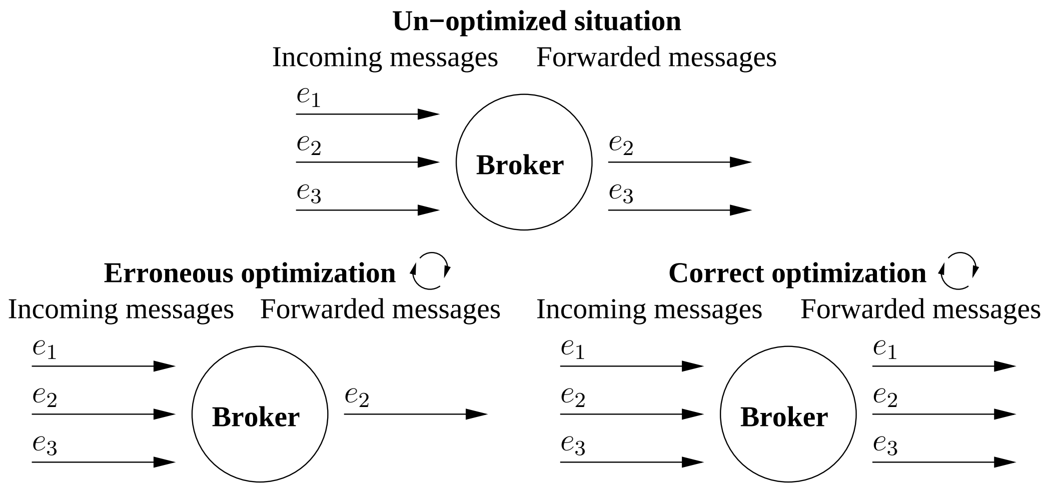


Figure 6.1: Schematic overview of an erroneous (bottom left) and a correct optimization (bottom right) of the un-optimized situation given at the top: the un-optimized situation forwards events e_2 and e_3 to a certain neighbor broker, whereas an erroneous optimization only forwards e_2 and a correct optimization e_1, e_2, and e_3.

6.1.1 Generalizing Subscriptions

Entries in event routing tables state what event messages will be forwarded to what subscribers and neighbors in the network. A manipulation of these entries thus changes this forwarding of event messages. In order to allow for correct event filtering, an optimization should only manipulate routing entries of non-local subscribers, that is, entries that determine the forwarding of messages to neighbor brokers. The notification of subscribers in this case is always based on the original, unaltered entries and thus remains accurate.

The arbitrary alteration of routing entries might lead to an erroneous routing process. This situation, for example, occurs if messages are not forwarded to neighbor brokers after applying the optimization (false negatives). The additional forwarding of messages (false positives), on the other hand, does not erroneously influence the routing process, provided local subscriptions remain unaltered (as is our assumption). We give a schematic example of these two situations in Figure 6.1. We illustrate the forwarding of three example messages (e_1 to e_3) to a particular neighbor of the broker that is given in the figure.

This alteration property leads to our general optimization idea: one can achieve an optimization of the overall pub-sub system with respect to our design properties by generalizing non-local routing entries in event routing tables. The term “generalization” refers to the *selectivity* of subscriptions. We use a general definition of selectivity:

Definition 6.1 (Selectivity) *The selectivity $sel(s)$ of a subscription s is the ratio of filtered event messages that match subscription s and the overall number of filtered messages when assuming a large overall number of filtered messages.*

The generalization of subscriptions might be achieved in various ways. We consider two approaches: predicate replacement, which is merely based on the predicates of subscriptions (Section 6.2), and subscription pruning, altering the syntactical structure of subscriptions (Section 6.3). Looking at this generalization concept in terms of existing pub-sub terms means that the more general subscription (i.e., the optimized subscription) covers the more restricted subscription (i.e., the un-optimized subscription). The generalization can thus be seen by the means of the event messages $\mathcal{E}(s)$ that match subscriptions s.

In the following two sections, we introduce two ways of generalizing subscriptions.

6.2 Predicate Replacement

Our first approach, *predicate replacement*, considers only the predicates that are utilized by a subscription. The overall idea is as follows:

> Consider a general Boolean subscription $s_i = (T_i^n, \mathcal{F}_i)$ after the syntactical rewriting (i.e., all negations are shifted down into leaf nodes, see Section 4.2.1, page 101). To get a more general (or equally selective) subscription s_j, one can replace any predicate $p_k \in \mathcal{P}(\mathcal{F}_i)$ by a more general predicate p_l.

Proceeding in this way results in a more general subscription s_j: for the two predicates, it holds $\mathcal{E}(p_k) \subseteq \mathcal{E}(p_l)$. For the sake of simplicity, we firstly consider a conjunctive subscription s_i. When replacing p_k by p_l, it holds that:

$$\mathcal{E}(s_i) = \bigcap_{p_m \in \mathcal{F}_i} \mathcal{E}(p_m) \quad \subseteq \quad \mathcal{E}(s_j) = \bigcap_{p_n \in (\mathcal{F}_i \setminus \{p_k\}) \cup \{p_l\}} \mathcal{E}(p_n).$$

Thus, s_j is more general after the replacement performed on s_i. Secondly, let us consider a disjunctive subscription s_i. When replacing p_k by p_l, it holds that:

$$\mathcal{E}(s_i) = \bigcup_{p_m \in \mathcal{F}_i} \mathcal{E}(p_m) \quad \subseteq \quad \mathcal{E}(s_j) = \bigcup_{p_n \in (\mathcal{F}_i \setminus \{p_k\}) \cup \{p_l\}} \mathcal{E}(p_n).$$

Hence, after performing a replacement on a disjunctive subscription s_i, the resulting subscription s_j is more general, that is, less or equally selective. Based on these two basic cases, the predicate replacement approach leads to more general (general) Boolean subscriptions. That is, predicate replacement is applicable for general Boolean subscriptions in the way we defined them (after the syntactical rewriting).

The determination of more general predicates can be based on the covering properties among predicates. These properties, in turn, can be derived from the operators that are used in predicates. Examples for covering properties based on operators in predicates are given in [Müh01].

6.2.1 Optimization Effects

Predicate replacement alters the predicate indexes that are used by a filtering algorithm. In order to achieve an optimization, one should replace all occurrences of a certain predicate p by a more general predicate. Proceeding in this way allows for the removal of this predicate p from predicate index structures (provided p is not shared by local subscriptions).

The removal of predicates from index structures, firstly reduces the memory requirements for these predicate indexes. Secondly, these indexes can be evaluated more efficiently. However, the size and general structure of subscription indexes remains unchanged by the application of predicate replacement. This is the reason for the development of a more advanced generalization method in Section 6.3.

We give an example of predicate replacement in Figure 6.2. Here we apply the optimization to node n_4 and node n_5 of subscription s_1 (see Section 3.3.1 on page 80). Assuming the replacement of all instances of the original predicates `Price` $<$ `NZ$15.00` and `Price` $<$ `NZ$12.00`, they can be removed from predicate indexes.

In the following subsection, we relate the predicate replacement approach to our design goals for an optimization.

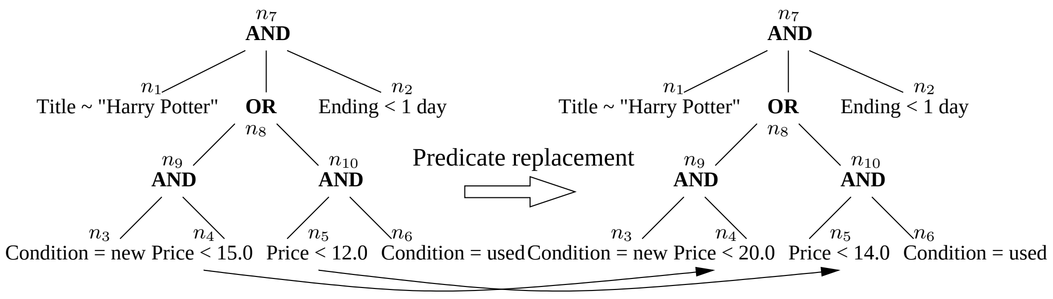


Figure 6.2: Example of predicate replacement on subscription s_1 when replacing `Price` $<$ `NZ$15.00` by `Price` $<$ `NZ$20.00` (Node n_4), and `Price` $<$ `NZ$12.00` by `Price` $<$ `NZ$14.00` (Node n_5).

6.2.2 Relation to Design Goals

Returning to the types of routing optimizations we identified, predicate replacement classifies as interfering routing optimization. Its target (primary) parameters are memory requirements and system efficiency. Additionally, the generalization of routing entries increases the internal network load (secondary parameter) because more event messages might match a more general subscription.

Regarding Design Goal 1, predicate replacement is applicable to general Boolean subscriptions, involving both conjunctions and disjunctions.

Furthermore, predicate replacement does not depend on the covering relationships among subscriptions (Design Goal 2). However, it assumes covering relationships among predicates. Otherwise one cannot replace predicates by more general ones in order to decrease the memory requirements for event routing tables. The assumption of these relationships is much weaker for predicates than for subscriptions, largely complying with Design Goal 2.

In the case of deregistrations, predicate replacement improves this process because predicate indexes contain fewer entries, resulting in a more efficient predicate removal (Design Goal 3). In the distributed system, the deregistration remains unaltered compared to the un-optimized situation.

The potential of predicate replacement to improve the time efficiency of event filtering (Design Goal 4) does exist, but only on an inferior basis. The reason for this is that only predicate indexes are altered by the replacement optimization. The proportion of filter time spent on their evaluation, however, is much less than the time for subscription matching, although the support of complicated filter functions in predicates increases the proportion of filter time spent for predicate matching. However, more general subscriptions lead to more forwarded messages. These messages also need to be routed by intermediate brokers (on the path to the subscriber), counteracting the improvement in efficiency.

Finally, Design Goal 5 is only suboptimally targeted by predicate replacement. Once more the reason for this effect is the alteration of predicate indexes, whereas subscription indexes remain unchanged. This particularly happens if the requirement that removed predicates are only used in non-local subscriptions (and not in local ones) does not hold.

Setting these properties of predicate replacement into perspective, we design a more suitable event routing optimization in the next section, that is, an optimization that better fulfills our design goals.

6.3 Subscription Pruning

Our second optimization proposal is *subscription pruning*, which considers the structure of subscriptions when generalizing Boolean filter expressions. We integrated this optimization into BoP. The general optimization idea is as follows:

> Let us consider a general Boolean subscription s_i after the syntactical rewriting (i.e., all negations are shifted down into leaf nodes, see Section 4.2.1). To get a more general (or equally selective) subscription s_j, one can remove, that is, *prune*, selected branches of the subscription tree of s_i.

Inner nodes of subscription trees after the syntactical rewriting contain disjunctions and conjunctions, whereas leaf nodes contain predicates. Candidates for pruning operations are thus the removal of a child of a disjunctive node and of a conjunctive node. However, only the latter option leads to a more general subscription, our requirement for an optimization:

Similarly to predicate replacement, let us firstly consider a pure conjunctive subscription s_i: when removing a predicate p_k (or, more generally, a child node), leading to subscription s_j, it holds that:

$$\mathcal{E}(s_i) = \bigcap_{p_m \in \mathcal{F}_i} \mathcal{E}(p_m) \subseteq \mathcal{E}(s_j) = \bigcap_{p_m \in \mathcal{F}_i \setminus \{p_k\}} \mathcal{E}(p_m).$$

Thus s_j becomes more general after the performed pruning operation. Secondly, let us consider a disjunctive subscription s_i and the removal of a predi-

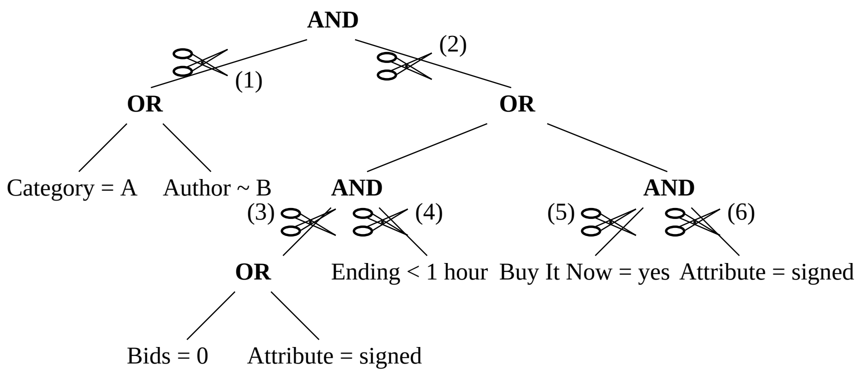


Figure 6.3: Overview of the six possible pruning operations for Subscription Class 3. The operations are named by (1) to (6).

cate p_k (or, more generally, a child node), leading to s_j. It now holds that:

$$\mathcal{E}(s_i) = \bigcup_{p_m \in \mathcal{F}_i} \mathcal{E}(p_m) \;\supseteq\; \mathcal{E}(s_j) = \bigcup_{p_m \in \mathcal{F}_i \setminus \{p_k\}} \mathcal{E}(p_m).$$

Hence the removal of a child of a disjunctive node might lead to an equally or more selective subscription. This pruning operation does not fulfill our requirement of creating a more general filter expression.

We can again derive the properties of pruning of general Boolean subscriptions from these two basic cases. That is, given a general Boolean subscription according to our definition (after the syntactical rewriting), any removal of a child of a conjunctive node leads to a more general subscription and is thus a valid pruning operation. Note that valid pruning operations can remove leaf nodes and disjunctive nodes (conjunctive nodes would be integrated into the parent node while preprocessing, see Section 4.2.1, page 101).

6.3.1 Post-processing

In Figure 6.3, we give an overview of all six possible pruning operations for subscriptions of Subscription Class 3. Having performed any of these operations in practice, BoP post-processes pruned subscriptions for compacting purposes.

Unary Operator Removal

Firstly, BoP applies the *unary operator removal* method to eliminate those inner nodes of subscription trees that have only one remaining child. In our example in Figure 6.3, after performing Pruning Option 6 the rightmost conjunctive node is removed by this method, leading to predicate `Buy It Now` = `yes` being the child of the rightmost disjunction. Pruning Option 6 thus eliminates a leaf (immediately) and an inner node (in post-processing).

Operator Summarization

In a second post-processing step, BoP again employs the operator summarization method that is already used in the preprocessing step before indexing subscriptions (see Section 4.2.1). For example, when performing Pruning Option 4 (in our example in Figure 6.3), BoP would initially remove the unary conjunction (parent of removed predicate) and afterwards summarize the two, now consecutive, disjunctions. The removal of predicate `Ending Within` < `1 hour` thus removes one leaf node (immediately) and two inner nodes (one due to unary operator removal and one due to operator summarization), reducing the encoding size of the subscription tree (see Section 4.2.2, page 102) from 47 to 40 bytes.

6.3.2 Optimization Effects

Subscription pruning alters the subscription indexes in pub-sub systems and thus the entries in event routing tables. The pruning approach might also affect the applied predicate indexes, provided all instances of a certain predicate are pruned.

Whereas existing optimizations aim at reducing the problem size of the event routing task, subscription pruning reduces the complexity of the event routing task by altering routing entries themselves. This property allows for the combination of pruning and recent optimizations, as we discuss in Section 6.3.5 and empirically show in Chapter 8.

There are two main effects on the pub-sub system if applying subscription pruning:

1. Subscriptions (i.e., routing entries) become less complex after performing pruning. Hence, in individual broker components, the routing load per event message decreases.

2. Subscriptions get more general due to pruning. This property increases the internal network load (due to false positives) among broker components. A consequence is the routing of more events to neighbor brokers

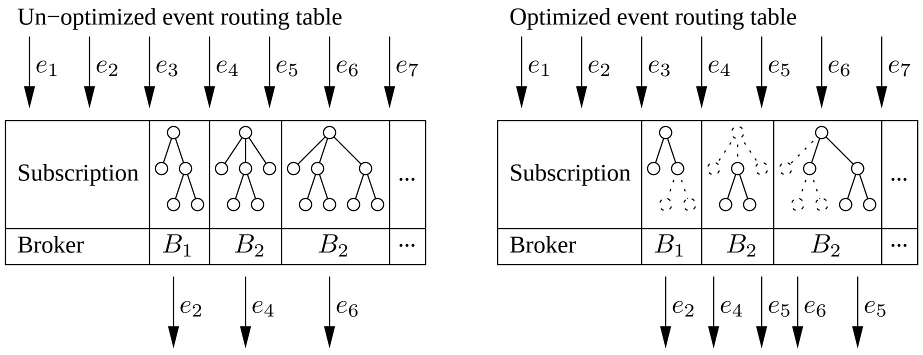

Un−optimized event routing table
e_1 e_2 e_3 e_4 e_5 e_6 e_7
Subscription
...
Broker
B_1
B_2
B_2
e_2 e_4 e_6
Optimized event routing table
e_1 e_2 e_3 e_4 e_5 e_6 e_7
Subscription
Broker
B_1
B_2
B_2
e_2 e_4 e_5 e_6 e_5

Figure 6.4: Example of the influences of subscription pruning. The dashed parts of subscription trees were pruned. The un-optimized setting (left) forwards two messages to broker B_2, whereas the optimized setting (right) forwards three messages to B_2. Message e_5 matches both pruned routing entries for B_2 in this example.

in the network. This growing internal network load triggers an increase in the number of event messages that brokers have to route compared to the un-optimized situation.

Regarding system efficiency, the second effect, that is, the routing of more event messages to brokers, counteracts the advantage of creating less complex routing entries (first effect). If the created, more general subscriptions lead to an introduction of various false positives, this negative effect of subscription pruning might outweigh the advantage of routing based on less complex subscriptions. However, if the number of false positives remains reasonable, the positive effect of less complex routing entries outbalances the drawback of routing more messages.

In practice, we expect an increasing routing efficiency up to a certain number of pruning operations. However, if performing a large amount of pruning, one might introduce too many false positives, leading to an overall decrease in system efficiency. The exact number of pruning operations one can perform to lead to improved filter efficiency depends on the structure of subscriptions, the application domain, and the pruning strategy (see Section 6.4).

Next to affecting system efficiency, subscription pruning always leads to a reduction in the memory requirements for subscription indexes and, potentially, for predicate indexes as well. Thus, at all times, one can apply subscription pruning to minimize the sizes of event routing tables.

We illustrate the effects of subscription pruning in Figure 6.4. From the seven event messages (e_1 to e_7) that are processed in the un-optimized setting (left part of the figure), one is forwarded to broker B_1 (e_2) and two are forwarded to B_2 (e_4 and e_6). After applying pruning (right part of the figure), firstly, routing entries get less complex (dashed parts were pruned), and secondly, three event message are now forwarded to B_2 (the false positive e_5 matches both pruned subscriptions in this example). Broker B_2 thus needs to filter more messages than in the un-optimized case. However, its routing entries become more efficient to evaluate (and store).

6.3.3 Relation to Design Goals

Similarly to predicate replacement, subscription pruning classifies as interfering subscription-based routing optimization. Target parameters are again the memory requirements for routing tables and the system efficiency. The internal network load is also affected by the application of pruning (as secondary parameter), leading to the classification of this approach.

Subscription pruning is applicable to general Boolean subscriptions (in the way we defined them). It thus fulfills Design Goal 1 (see Section 6.1). Although pure disjunctive subscriptions cannot be optimized by our pruning approach, its applicability in practice is not undermined by this fact: we are not aware of any applications that require mere disjunctive subscriptions[1].

Subscription pruning is independent of the covering relationships among the registered subscriptions (Design Goal 2). We empirically show this property in Chapter 8. By applying pruning, the existing covering relationships in the system might be altered. If pruning does not change the covering relationships among subscriptions from the same neighbor broker, the optimization does not increase the network load in the system.

Design Goal 3, the non-negative influence on deregistrations, is also fully fulfilled by subscription pruning. Deregistrations are more positively influenced by subscription pruning than by predicate replacement because subscription index structures are strongly reduced in their complexity when applying pruning. Hence the removal of subscriptions is supported more efficiently than in the un-optimized setting (due to the smaller size of these indexes). We refer to Section 6.6.3 for details on the handling of deregistrations in BoP.

We already discussed the twofold influence of subscription pruning on system efficiency (Design Goal 4) in Section 6.3.2. We show that its advantages

[1]If such applications do exist, one should rather apply predicate replacement.

outweigh its disadvantages in Chapter 8, and can hence state that pruning increases filter efficiency.

Finally, Design Goal 5 is fulfilled by subscription pruning because each performed pruning operation reduces the size of event routing tables. Hence the pub-sub system can integrate more routing entries for a given amount of main-memory resource when applying subscription pruning.

6.3.4 Connection between Subscription Pruning and Predicate Replacement

Although, at a first glance, subscription pruning and predicate replacement appear as different generalization approaches for subscriptions, one could look at them as the same overall approach. Alternatively, there is potential to combine both ideas.

Treating Subscription Pruning as Specialized Predicate Replacement

An abstract view on the introduced predicate replacement and subscription pruning methods reveals their interconnection: subscription pruning could be looked at as a specialized form of predicate replacement, followed by a post-processing:

Pruning is equivalent to the replacement of predicates or whole branches of subscription trees by the most general predicate p_*, which is fulfilled by each event message (similar to "don't-care" predicates, see Section 4.7, page 116). Certain branches might then be removed from the tree structure because they do not restrict fulfilling event messages. This replacement and removal does lead to a more general filter expression for the children of conjunctive nodes, as described by our pruning rule.

Removing a "don't-care" predicate p_* as a child of a disjunction, however, does not lead to a more general subscription. For example, the replacement of `Author` $\sim$ `B` by p_* in Figure 6.3 (page 156) means that the leftmost disjunction of this subscription is always fulfilled. A subsequent removal of p_* then restricts the subscription again.

However, one could remove this whole disjunctive branch, which is described by Pruning Option 1. Our single pruning rule hence describes the replacement of any predicate and the correct post-processing when considering pruning as a specialized predicate replacement variant.

Combining Subscription Pruning and Predicate Replacement

As argued in the previous subsection, subscription pruning might be seen as a specialized predicate replacement variant in combination with a (simple) semantic post-processing. However, according to Section 6.2, the original, that is, non-postprocessing, predicate replacement strategy has less optimization potential than subscription pruning. When combining both generalization strategies, though, this joint optimization might have a higher optimization effect than their individual application.

Despite this option, we only consider the pure subscription pruning optimization in the following. Our decision to proceed in this way is based on the overall larger optimization potential of subscription pruning compared to predicate replacement. Additionally, predicate replacement only reduces the memory requirements if all occurrences of a predicate are removed, which is difficult to realize in practice. Finally, predicate replacement involves a higher computational effort than subscription pruning [BH06d].

6.3.5 Pruning and Existing Optimizations

In contrast to the covering, merging, and summarization optimization approaches, aiming at decreasing the number of routing table entries, subscription pruning targets the reduction of the complexity of routing entries. This allows for the combination of pruning and current approaches due to these opposing dimensions of optimization:

Brokers should, for example, exploit the cover among subscriptions to remove redundant routing entries. Additionally, they can apply subscription pruning to reduce the complexity of the remaining entries. Thus, next to its own positive effects on the routing process, subscription pruning can exploit the benefits of other optimizations due to the opportunity to combine it with these approaches. We show this behavior for the auctioning example dataset in Chapter 8.

Utilizing Pruning for Imperfect Merging

One can also utilize subscription pruning to create an imperfect merger for general Boolean subscriptions. This overcomes the restrictions of all current merging approaches, which only work on conjunctions.

In order to find an imperfect merger, one firstly creates a perfect merger by building the disjunction of all subscriptions to be merged. This results in a subscription that summarizes all merged filter expressions. Secondly, one prunes the created merger. With each pruning step, this merger becomes more inaccurate, that is, imperfect. The number of performed pruning operations determines the degree of imperfectness of the merger.

This approach automatically overcomes the problem of deciding how to merge subscriptions [LHJ05], provided one has determined a pruning strategy. In the following section, we present several strategies to select the preferable pruning option to perform. When using an approach that is based on the selectivity of subscriptions, the imperfect merger is expected to be relatively accurate for its size but less complex than the original perfect merger. That is, one can evaluate the imperfect merger more efficiently than its merged constituents, this being one goal of the merging optimization.

6.4 Selecting Pruning Operations

An important open question remains for the application of subscription pruning: if the pub-sub system has registered hundreds of thousands of subscriptions, how does it decide what pruning operations should be performed?

6.4.1 Ranking the Pruning Operations

Generally whenever a subscription s is registered with the pub-sub system, the system analyzes s. The result of this analysis leads to *local optimization* decisions: each subscription is ranked with a numeric value, stating the optimization potential for its preferred pruning operation. The local optimization thus determines the preferred pruning among all possible options for one subscription.

Having obtained this knowledge, the pub-sub system can easily reach *global optimization* decisions whenever an optimization is required. These decisions are based on the local rankings of subscriptions. They do not involve the complex relation of all subscriptions with each other, as required by current approaches, but only a comparison of local rankings.

In the following subsections, we propose several measures $\Delta(s_i, s_j)$ to rank the pruning operations of a subscription s_i that is pruned to s_j [BH06b]. These measures aim at optimizing the pub-sub system with respect to various target parameters and have been integrated into BoP.

6.4.2 Pruning Based on Subscription Accuracy

The *accuracy measure* ranks pruning operations based on their influence on the accuracy, that is, the selectivity (see Definition 6.1, page 152), of subscriptions. To a certain extent, the change (decrease) in selectivity when pruning subscriptions determines the number of false positives, and thus the increase in internal network load and routed messages. This direct influence only occurs if subscriptions do not overlap (we here refer to the overlap among subscriptions, i.e., subscription-subscription overlapping, and not among subscription and advertisements, as defined in Section 2.1.3). If subscriptions from the same neighbor broker overlap, the change in accuracy cannot be directly related to the increase in network load. We discuss this behavior later in this section. Prior to this, we introduce our means of determining the selectivity of subscriptions and of calculating the change in selectivity when pruning.

Estimating the Selectivity

Our proposal to express the selectivity of a subscription s uses an estimation $sel^{\approx}(s)$ for the actual selectivity $sel(s)$ of s. Taking this approach allows for the time- and space-efficient approximation of the subscription selectivity for general Boolean expressions. We refer to Section 6.7.2 for an overview of related selectivity calculation approaches that require the conversion to canonical forms, being exponential in size in the worst case.

Our selectivity estimation is based on three components, describing the minimal possible selectivity, the average selectivity, and the maximal possible selectivity of a subscription s: $sel^{\approx}(s) = (sel^{min}(s), sel^{avg}(s), sel^{max}(s))$. The minimal possible selectivity describes the worst case, that is, the smallest value of selectivity that holds for all possible distributions of event messages; the average case assumes a uniform distribution of all possible event messages and independent predicates in subscriptions; the best case is described by the maximal possible selectivity, that is, the largest selectivity value for any distribution of event messages. One can base the calculation of these components solely on the selectivity of predicates for any Boolean filter expression. In turn, this predicate selectivity can be derived from historic information and interpolation.

Determining Predicate Selectivity. Based on our selectivity definition, the selectivity of a predicate p, $sel(p)$, is the ratio of filtered event messages that fulfill p, $|msg_f(p)|$, and the total number of filtered messages, $|msg_t|$: $sel(p) = \frac{|msg_f(p)|}{|msg_t|}$ with $0 \leq sel(p) \leq 1$. This selectivity value holds for all three estimation components, that is, our estimation for predicates is the actual predicate selectivity.

BoP derives this selectivity based on historic information by keeping a counter per registered (unique) predicate p. This counter is increased whenever p is fulfilled by an incoming message. BoP also knows about the total number of filtered messages $|msg_t|$ and can thus calculate $sel(p)$ for any registered predicate p. Periodically recalculating $|msg_f(p)|$ allows a system to adapt to changes in event messages.

Taking this approach for newly registered predicates requires a start-up period to allow for a relatively stable selectivity value (some messages need to be filtered). This start-up period is not problematic because an optimization for newly registered subscriptions is not instantly required.

However, one can initially approximate the selectivity of newly registered predicates based on existing information about already registered predicates. A similar approach is presented in [GR03], using the selectivity of covering or covered subscriptions to estimate the selectivity for newly registered subscriptions.

For our problem, an approximation only needs to be undertaken for predicates. One can base this approximation on the filter functions used in predicates, as shown in the following example. However, sophisticated filter functions can lead to a more complex computation process, and the mere registration of unrelated predicates does not allow for relatively accurate approximations.

Example 6.1 (Selectivity approximation) *Let us consider the three filter functions for the comparison operators $=$, $>$, and $<$.*

- *To approximate the selectivity for a predicate $p_=$, one firstly needs to find the predicate $p_{>g}$ with the greatest operand that uses function "$>$" and covers $p_=$. Secondly, one determines the predicate $p_{>s}$ with the smallest operand that uses function "$>$" and does not cover $p_=$. The difference between the selectivity of these two predicates states the selectivity of all domain values between the operands of $p_{>s}$ and $p_{>g}$. One can thus approximate the selectivity of the value used in $p_=$, which can be further narrowed down if the selectivity of some of the values in this interval is* *known. Similarly, one can base this calculation solely on filter function "<", or on a combination of functions "<" and ">".*

- *To approximate the selectivity for a predicate $p_>$, one needs to find two predicates that use the same function: $p_{>g}$ specifies the greatest operand and covers $p_>$, and $p_{>s}$ specifies the smallest operand that does not cover $p_>$ anymore. Considering the selectivity of $p_{>s}$, and the respective proportion of the difference between the selectivity of $p_{>s}$ and $p_{>g}$ leads to a selectivity approximation for $p_>$. A known selectivity of equality predicates with operands between those of $p_{>s}$ and $p_{>g}$ further confines the approximation. This procedure works similarly for predicates only involving "<", or a combination of "<" and ">".*
- *The selectivity approximation for predicates involving function "<" works analogously to the previous case.*

Such an approximation process is not necessarily required in practice: an optimization is only needed if a large number of subscriptions is registered with the system or the frequency of event messages is high. In this case, the system has already processed a large number of messages and can thus optimize based on accurate information about the selectivity of registered predicates. Subscriptions that are registered after the optimization process can then be considered in the next optimization step, having left enough time to collect accurate selectivity information. Thus accurate selectivity values are always known at the time of optimization.

We give the pseudo code for this selectivity calculation approach in Algorithm 2. The algorithm expects a leaf node n as input parameter, and calculates the selectivity as a ratio of messages matching the predicate in n (function MatchingMessages()) and total number of filtered messages (function TotalMessages()).

```
Algorithm 2: Selectivity estimation for leaf nodes
Input: A leaf node $n$
Output: Estimation $sel^{\approx}(n) = (sel^{min}(n), sel^{avg}(n), sel^{max}(n))$
EstimateSelectivity(n)
(1)     est ← MatchingMessages(n) ÷ TotalMessages()
(2) return (est, est, est)
```

Deriving Subscription Selectivity. Based on the selectivity information of predicates, the system can recursively estimate the selectivity of any general Boolean subscription, that is, a filter expression that contains conjunctions and disjunctions as inner nodes.

For conjunctive nodes n_c of subscription trees, we present the pseudo code in Algorithm 3: the algorithm walks through all children of the input node (Line 4) and recursively estimates their selectivity (Line 5). The minimal selectivity value occurs if the sets of event messages that match the different children overlap (i.e., intersect) minimally (Line 6). Finally, the calculation result needs to be adjusted to be not less than zero (Line 9). The average selectivity value, according to our independence assumption, presumes that the derived selectivity of one child equally holds for the messages that match the other children (Line 7) of node n_c. For the maximal value, it is assumed that the matching messages of all children are included in the set of event messages that match the least selective child of node n_c (Line 8).

Algorithm 3: Selectivity estimation for conjunctive nodes
Input: A conjunctive node n
Output: Estimation $sel^{\approx}(n) = (sel^{min}(n), sel^{avg}(n), sel^{max}(n))$

```
EstimateSelectivity(n)
(1)  min ← 1.0
(2)  avg ← 1.0
(3)  max ← 1.0
(4)  foreach c in n.children
(5)      e ← EstimateSelectivity(c)
(6)      min ← min + e.min − 1.0
(7)      avg ← avg × e.avg
(8)      max ← Min(max, e.max)
(9)  if min < 0.0
(10)     min ← 0.0
(11) return (min, avg, max)
```

Algorithm 4 illustrates the calculation for disjunctive nodes n_d of subscription trees. Once more, the estimation algorithm walks through all children of input node n_d (Line 4) and recursively estimates their selectivity (Line 5).

For the minimal selectivity value, the algorithm needs to assume that the sets of matching event messages of all children are included in the largest of these sets (Line 6). Meeting our assumptions for the average selectivity, this case presumes uniformly distributed event messages and independence between branches of subscription trees (Line 7).

The maximal selectivity value occurs if the sets of matching event messages of all children of disjunctive node n_d are maximally disjoint (Line 8). Finally, the calculation result is corrected to its maximum value of 1.0 (Line 9).

Algorithm 4: Selectivity estimation for disjunctive nodes
Input: A disjunctive node n
Output: Estimation $sel^{\approx}(n) = (sel^{min}(n), sel^{avg}(n), sel^{max}(n))$

```
EstimateSelectivity(n)
(1)   min ← 0.0
(2)   avg ← 0.0
(3)   max ← 0.0
(4)   foreach c in n.children
(5)      e ← EstimateSelectivity(c)
(6)      min ← Max(min, e.min)
(7)      avg ← avg + e.avg − (avg × e.avg)
(8)      max ← max + e.max
(9)   if max > 1.0
(10)     max ← 1.0
(11)  return (min, avg, max)
```

Having these means to estimate the selectivity for any nodes of subscription trees, the selectivity estimation for a subscription s equals the estimation for its root node n, that is, $sel^{\approx}(s) = sel^{\approx}(n)$.

Selectivity Degradation Measure

We started this section with the fundamental question of how to select pruning operations. Being able to estimate the selectivity of any subscription allows us to quantify the effect of pruning operations and to rank them according to their influence on selectivity. This ranking can be obtained in two ways, either from the absolute change in selectivity or the proportional change.

We favor an absolute measure because it more accurately relates the decrease in selectivity due to pruning to the expected increase in network load. A proportional measure, on the other hand, weights the selectivity decrease according to the original selectivity of the unpruned subscription. That is, if a relatively general subscription is pruned, the proportional change in selectivity is smaller than the change for a quite restrictive one. However, the induced increase in network load might be much larger for the general subscription.

We define our absolute pruning measure, the *estimated selectivity degradation* $\Delta^{\approx}_{sel}(s_i, s_j)$, as the maximal difference between the selectivity estimation components of a subscription s_i before pruning and after pruning to s_j:

$$\begin{aligned}\Delta_{sel}^{\approx}(s_i, s_j) \quad = \max(\quad & sel^{min}(s_j) - sel^{min}(s_i), \\ & sel^{avg}(s_j) - sel^{avg}(s_i), \\ & sel^{max}(s_j) - sel^{max}(s_i)).\end{aligned}$$

Having defined this degradation measure, a pub-sub system can reach local optimization decisions by determining the pruning operation (leading to s_j) for subscription s_i that leads to the smallest estimated selectivity degradation $\Delta_{sel}^{\approx}(s_i, s_j)$. Out of these local decisions and their ordering by the degradation measure, the system then comes to global optimization decisions.

When performing several pruning operations on one subscription in a row, s_i always refers to the original subscription. Proceeding in that way allows for the incorporation of the overall change in selectivity of a subscription in our degradation measure $\Delta_{sel}^{\approx}(s_i, s_j)$. When using an already-pruned subscription for this measure, several small degradation values as the result of continuously pruning s_i might appear as a reasonable choice. However, adding up these individual values reveals the total degradation after pruning. Our approach of choosing the unpruned subscription s_i for calculations avoids this problem and always represents the overall effect of the optimization rather than the additional effect compared to the pruning operation performed before.

Validity of the Degradation Measure

The overall goal of our selectivity degradation measure is to order pruning operations according to their influence on the network load. However, there are differences between the selectivity degradation and the increase in network load:

1. A decrease in selectivity does not necessarily result in an increased network load, that is, false positives.
2. Our selectivity estimation, as the name suggests, is only an approximation of the real selectivity of subscriptions.
3. The selectivity degradation measure, firstly, is based on an estimation and, secondly, does not model the worst case degradation.

Difference 1 results from the fact that subscribers usually do not specify an exclusive interest in particular event messages. One can expect that subscriptions overlap partially (or completely, which is described by the covering relationship): if a performed pruning operation on a subscription s_i leads to false positives, these additional messages might already be described by at least one other subscription s_j that was forwarded by the same neighbor broker as s_i. These additional messages are thus not false positives from a global, or broker-wise, viewpoint.

Whether pruning operations lead to the number of false positives that is predicted by the degradation measure depends on the overlapping properties among subscriptions. The amount of cover among subscriptions contributes to these properties. In Chapter 8 we show the performance of the selectivity degradation measure for various covering proportions. As we will see, subscription pruning is applicable regardless of the cover among subscriptions.

To incorporate the overlap among subscriptions into an optimization, one needs to relate (all) subscriptions to each other. Such an approach is too time-consuming in the context of pub-sub systems involving general Boolean expressions, and has clearly been identified as one of the drawbacks of current optimization approaches (see Section 2.5, page 47). We thus do not take such a step within this dissertation, although we do develop a refined degradation measure that partially incorporates the relationships among subscriptions into our pruning optimization. Nevertheless, the selectivity degradation measure presented here fulfills its goal of considering the increase in network load, as we empirically show in Chapter 8.

Difference 2 describes the property of determining the selectivity of a subscription s. It does not calculate its real selectivity $sel(s)$ but only estimates it by $sel^{\approx}(s)$. This estimation provides us with an interval in which the selectivity can be found. It always holds that: $sel^{min}(s) \leq sel(s) \leq sel^{max}(s)$. The third estimation component, $sel^{avg}(s)$, describes an expected selectivity value, provided certain assumptions hold. It thus states which of the two extremes is more likely, for example, when assuming independent predicates. Obviously such an approximation does not accurately model the real selectivity $sel(s)$ under all circumstances. It introduces a certain fuzziness into the selectivity values of subscriptions.

This fuzziness directly leads to Difference 3: if the base concept, the selectivity of a subscription, is an estimation, the derived concept, the difference in selectivity, cannot become an exact notion. Additionally, our selectivity degradation measure does not represent the worst case change in selectivity

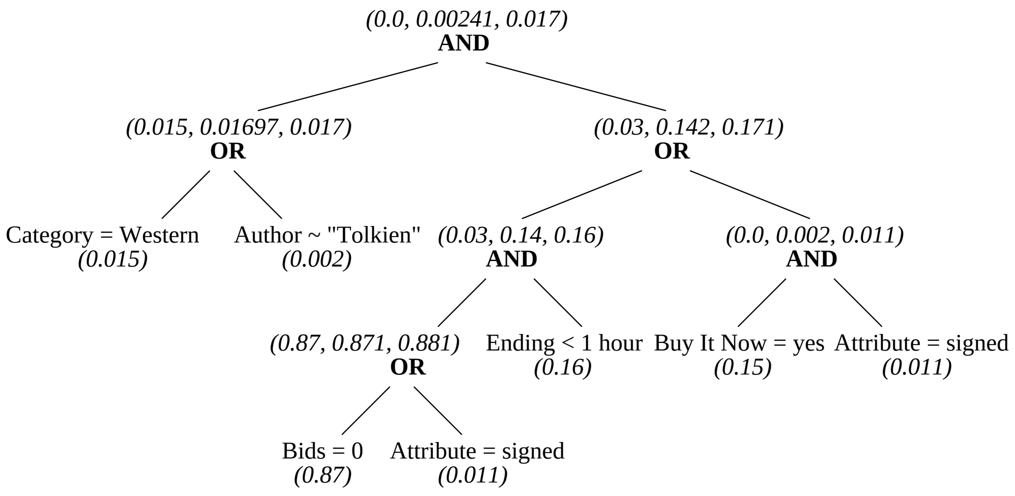


Figure 6.5: Selectivity estimation for a subscription of Subscription Class 3.

when performing pruning. This worst case when pruning subscriptions s_i to s_j is: $\Delta_{sel}^{max}(s_i, s_j) = sel^{max}(s_j) - sel^{min}(s_i)$.

Despite this fuzziness of our general selectivity notion and its degradation, in practice the presented concepts lead to an effective optimization, as shown in Chapter 8.

Properties of Example Subscription Class

Having theoretically introduced our selectivity estimation, we now present a calculation example using Subscription Class 3:

Example 6.2 (Selectivity estimation) *In Figure 6.5, we illustrate a subscription of Subscription Class 3. At the leaf level of the subscription tree, we give the selectivity of the predicates that are stored in leaf nodes. We show the selectivity value only once because all three estimation components are the same for leaf nodes: the selectivity of the stored predicate, either merely determined with the help of historic information or in combination with the approximation method.*

The recursive estimation of the selectivity of this subscription works upwards to the root node. For example, for the rightmost conjunctive node n_c (two levels down from the root), it holds in the first iteration of Algorithm 3:

$$\begin{aligned} sel^{min}(n_c) &= 1.0 + 0.15 - 1.0 = 0.15 \ (\mathit{Line\ 6}), \\ sel^{avg}(n_c) &= 1.0 \times 0.15 = 0.15 \ (\mathit{Line\ 7}), \\ sel^{max}(n_c) &= \min(1.0, 0.15) = 0.15 \ (\mathit{Line\ 8}). \end{aligned}$$

The second and final iteration of Algorithm 3 leads to the selectivity estimation of n_c, as shown in the figure:

$$\begin{aligned}
sel^{min}(n_c) &= 0.15 + 0.011 - 1.0 = -0.839 \ (\textit{Line 6}) = 0 \ (\textit{Line 9}),\\
sel^{avg}(n_c) &= 0.15 \times 0.011 \approx 0.002 \ (\textit{Line 7}),\\
sel^{max}(n_c) &= \min(0.15, 0.011) = 0.011 \ (\textit{Line 8}).
\end{aligned}$$

Subsequently, for the rightmost disjunction n_d (right child of the root), the first iteration of Algorithm 4 leads to:

$$\begin{aligned}
sel^{min}(n_d) &= \max(0.0, 0.03) = 0.03 \ (\textit{Line 6}),\\
sel^{avg}(n_d) &= 0.0 + 0.14 - (0.0 \times 0.14) = 0.14 \ (\textit{Line 7}),\\
sel^{max}(n_d) &= 0.0 + 0.16 = 0.16 \ (\textit{Line 8}).
\end{aligned}$$

The second and final iteration of Algorithm 4 then results in the estimation that is stated in Figure 6.5:

$$\begin{aligned}
sel^{min}(n_d) &= \max(0.03, 0.0) = 0.03 \ (\textit{Line 6}),\\
sel^{avg}(n_d) &= 0.14 + 0.002 - (0.14 \times 0.002) \approx 0.142 \ (\textit{Line 7}),\\
sel^{max}(n_d) &= 0.16 + 0.011 = 0.171 \ (\textit{Line 8}).
\end{aligned}$$

Finally, the calculation for the second iteration of Algorithm 3 for the root node n is as follows (we omit the first iteration due to its simplicity):

$$\begin{aligned}
sel^{min}(n) &= 0.015 + 0.03 - 1.0 = -0.955 \ (\textit{Line 6}) = 0 \ (\textit{Line 9}),\\
sel^{avg}(n) &= 0.01697 \times 0.142 \approx 0.00241 \ (\textit{Line 7}),\\
sel^{max}(n) &= \min(0.017, 0.171) = 0.017 \ (\textit{Line 8}).
\end{aligned}$$

Having estimated the selectivity of the subscription in Figure 6.5, we now demonstrate how to find the preferred pruning option (of the six possible options that are shown in Figure 6.3), using our selectivity degradation measure:

Example 6.3 (Selectivity degradation) *The unpruned selectivity estimation of subscription s_i in Figure 6.5 is $sel^{\approx}(s_i) = (0.0, 0.00241, 0.017)$ (see Example 6.2). After performing a pruning that leads to s_j, these estimations* *change to:*

$$\begin{aligned}
sel^{\approx}(s_j) &= (0.03, 0.142, 0.171) \textit{ for Pruning Option 1},\\
sel^{\approx}(s_j) &= (0.015, 0.01697, 0.017) \textit{ for Pruning Option 2},\\
sel^{\approx}(s_j) &= (0, 0.00275, 0.017) \textit{ for Pruning Option 3},\\
sel^{\approx}(s_j) &= (0, 0.0148, 0.017) \textit{ for Pruning Option 4},\\
sel^{\approx}(s_j) &= (0, 0.0026, 0.017) \textit{ for Pruning Option 5},\\
sel^{\approx}(s_j) &= (0, 0.0029, 0.017) \textit{ for Pruning Option 6}.
\end{aligned}$$

When applying the selectivity degradation measure, the rankings of the pruning options are (ordered by the option number):

$$\begin{aligned}
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.03 - 0.0, 0.142 - 0.00241, 0.171 - 0.017) = 0.154,\\
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.015 - 0.0, 0.01697 - 0.00241, 0.017 - 0.017) = 0.015,\\
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.0 - 0.0, 0.00275 - 0.00241, 0.017 - 0.017) = 0.00034,\\
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.0 - 0.0, 0.0148 - 0.00241, 0.017 - 0.017) = 0.01239,\\
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.0 - 0.0, 0.0026 - 0.00241, 0.017 - 0.017) = 0.00019,\\
\Delta^{\approx}_{sel}(s_i, s_j) &= \max(0.0 - 0.0, 0.0029 - 0.00241, 0.017 - 0.017) = 0.00049.
\end{aligned}$$

The preferred pruning option, that is, the pruning option leading to the least degradation, is Pruning Option 5 followed by Pruning Options 3, 6, 4, 2, and 1.

An informal reason for this decision is that Option 5 only broadens the subscription to include all signed copies of the books of a particular category or author. This pruning does not significantly change the selectivity because signed copies only constitute approximately 1 percent of all items. Even if no Buy-It-Now items would be signed book copies, the selectivity does maximally increase by this 1 percent.

The branch that is removed by Pruning Option 3 is fulfilled by approximately 87 percent of all messages. Because its sibling is fulfilled by 16 percent, the selectivity of the (then unary) conjunction can increase by 13 percent at most. For Pruning Option 6, the selectivity of the (then also unary) conjunction could increase by 15 percent. Applying Option 4, the average selectivity change of the conjunction (again, then unary) would be quite significant. Obviously, Pruning Option 2 generalizes more than any partial pruning (even though, the *degradation measure might lead to a different result). Finally, Pruning Option 1 would remove quite restrictive attributes on category and author, strongly broadening the overall subscription.*

6.4.3 Pruning Based on Filter Efficiency

Having proposed a measure that is based on the accuracy of subscriptions and aims at describing the increase in internal network load, we now introduce additional pruning variants that optimize with respect to other target parameters. In this section, we propose a ranking measure that aims at increasing filter efficiency as much as possible when pruning subscriptions—the *efficiency measure*.

An effective efficiency-based ranking measure should be intertwined with the applied filtering algorithm. Thus we use the approach that was presented in Chapter 4 as our target algorithm for the optimization.

Estimating the Effect on Filter Efficiency

The overall largest proportion of filter time in the general Boolean filtering algorithm is spent on final subscription matching. This third step of the algorithm evaluates the subscription trees of candidate subscriptions. Our approach to optimizing the filtering process is thus to minimize the additional number of candidate subscriptions that is created due to the performed pruning operations.

A main parameter that determines whether a registered subscription s, constitutes a candidate is the minimal number of fulfilled predicates $p_{min}(s)$. The pruning measure therefore aims at beneficially altering this subscription-specific property.

Due to the definition of $p_{min}(s)$ (see Section 4.2.3) and the options for valid pruning operations, the value of $p_{min}(s)$ cannot increase when pruning subscription s. Hence, $p_{min}(s)$ either increases or remains the same by the application of subscription pruning.

Although the filter efficiency is influenced by various parameters (as argued later on, page 175), for our ranking measure we consider the value of $p_{min}(s)$ as an estimation of the efficiency properties of subscriptions.

Efficiency Improvement Measure

If two subscriptions s_i and s_j share the same predicates and structure, the property $p_{min}(s_i) > p_{min}(s_j)$ generally means that subscription s_j is a candidate more often than ${s_i}^2$. To advantageously affect the filter efficiency of a subscription s, which is estimated by $p_{min}(s)$, the system should thus try to increase the value of $p_{min}(s)$ when pruning. However, an increase of this property is impossible when applying pruning. Thus, the system should rank pruning operations based on their decreasing effect on $p_{min}(s)$.

We define the efficiency-based ranking measure, the *efficiency improvement*, as the difference in $p_{min}(s)$ of a subscription s before and after pruning. More specifically, given a subscription s_i that is pruned to s_j, the efficiency improvement is defined as follows:

$$\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) \quad = \quad p_{min}(s_i) - p_{min}(s_j).$$

The less the efficiency improvement of a pruning operation, the more advantageously this operation influences filter efficiency. To reach local optimization decisions, pub-sub systems thus determine the pruning option that leads to the least ranking value for a given subscription. For global decisions, the system executes local decisions in an ascending order of this rank.

Subscription s_i in $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$ again refers to the original, unpruned subscription. This procedure allows for the incorporation of all pruning operations that are performed on s_i into the ranking measure. If a pruning causes a strong decrease in the minimal number of fulfilled predicates, this property is integrated into the ranking of all subsequently performed pruning operations. On the other hand, if each pruning is regarded individually, a repeated decrease of the ranking, for example, by value v, is weighted more beneficially than a one-time high decrease, for example, by $2 \times v$. However, the overall decrease in the ranking of a subscription is much smaller for the pruning option that involves the higher decrease (of $2 \times v$). It is thus important to incorporate the influence of all pruning operations of a subscription into its ranking, as implemented by our efficiency improvement measure.

[2]There are potential cases in which this statement is too strong because both subscriptions always constitute a candidate for the same messages. It should thus read "…s_j is a candidate not less often than s_i."

Validity of the Measure

The efficiency improvement measure $\Delta_{\mathit{eff}}^{\approx}(s_i, s_j)$ does not incorporate all influences of pruning operations on the efficiency of filtering algorithms. It only considers one of these effects, the minimal number of fulfilled predicates. However, there are some other influences on filter efficiency in general, and on the decision of being a candidate subscription in particular.

Firstly, the predicates of subscriptions determine the actual number of fulfilled predicates for an incoming event message, which is stored in the hit vector. The comparison of the entry in the hit vector to the minimally required number of fulfilled predicates (in the minimum predicate count vector) then states whether a subscription constitutes a candidate. The actual number of fulfilled predicates is thus a further influence on filter efficiency. The selectivity of the pruned branch of a subscription tree determines how often the predicate counter is increased due to the predicates of this branch.

If only considering the node n whose child is pruned (n is a conjunction), the removal of a highly selective child reduces $p_{min}(n)$ by at least one, although the entry in the hit vector is increased only rarely by the predicates in the pruned branch. Conversely, the removal of a relatively general child node prunes a branch that leads to more increases in the hit vector. It is hence more advantageous to prune a general child than a selective one because the implied decrease of $p_{min}(n)$ is counterbalanced by the relatively frequent increase in the hit vector before pruning. Thus the effect of pruning is less than in the case of removing highly selective branches. This influence of the selectivity of pruned branches is partially included in BoP due to its strategy of breaking ties in the efficiency improvement measure. We elaborate on this strategy in Section 6.4.8.

Secondly, another influence on filter efficiency regards the semantics of predicates in subscriptions. The influence of these semantics on the minimal number of fulfilled predicates is described in Section 4.5.5 (page 113). If applying the optimization described there, one should consider the effects of pruning on the potential of this approach. Additionally, pruning operations could aim at increasing $p_{min}(s)$ of the subscription s to be pruned by introducing those situations that offer an optimization potential.

Our estimation of the influence of pruning on filter efficiency results in a good approximation of these effects (see our results in Chapter 8). The measure thus fulfills its design goals and shows the feasibility to prune subscriptions, and thus to optimize content-based pub-sub systems, based on the efficiency parameter.

Decisions for Example Subscription Class

We now give an example of calculating the efficiency improvement measure $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$ for our example subscription of Class 3:

Example 6.4 (Efficiency improvement) *For our original subscription s_i, it holds $p_{min}(s_i) = 3$ for the minimal number of fulfilled predicates (see Example 4.6, page 109). Performing the six possible pruning operations (shown in Figure 6.3) that result in s_j leads to the following values:*

$$\begin{aligned}
p_{min}(s_j) &= 2 \textit{ for Pruning Option 1},\\
p_{min}(s_j) &= 1 \textit{ for Pruning Option 2},\\
p_{min}(s_j) &= 2 \textit{ for Pruning Option 3},\\
p_{min}(s_j) &= 2 \textit{ for Pruning Option 4},\\
p_{min}(s_j) &= 2 \textit{ for Pruning Option 5},\\
p_{min}(s_j) &= 2 \textit{ for Pruning Option 6}.
\end{aligned}$$

Using the efficiency improvement measure, the rankings of these pruning options are:

$$\begin{aligned}
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 2 = 1 \textit{ for Pruning Option 1},\\
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 1 = 2 \textit{ for Pruning Option 2},\\
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 2 = 1 \textit{ for Pruning Option 3},\\
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 2 = 1 \textit{ for Pruning Option 4},\\
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 2 = 1 \textit{ for Pruning Option 5},\\
\Delta^{\approx}_{\mathit{eff}}(s_i, s_j) &= 3 - 2 = 1 \textit{ for Pruning Option 6}.
\end{aligned}$$

The preferred pruning options, that is, the ones that are expected to improve filter efficiency the most, are Pruning Options 1, 3, 4, 5, and 6. Pruning Option 2, however, has a stronger (negative) influence on filter efficiency.

As mentioned in Section 6.4.3, BoP *applies an extended strategy to select the preferred pruning operation if various options result in the same efficiency improvement $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$. This strategy then leads to a definite decision for the tie among Pruning Options 1, 3, 4, 5, and 6.*

The reason for Pruning Option 2 to be rated as the worst pruning is that *only one predicate of the remaining subscription s_j has to be fulfilled to designate s_j as a candidate subscription.*

6.4.4 Pruning Based on Memory Usage

Having proposed measures for the parameter network load and the quality measure system efficiency, we now introduce the *memory measure* that primarily aims at decreasing the memory requirements for event routing tables.

We have already elaborated on the influence of subscription pruning on both predicate and subscription indexes. Its effect on subscription indexes was identified as more significant with respect to memory usage than its effect on predicate indexes. In the following, we thus approximate the memory requirements of an event routing entry (i.e., a subscription) by its size in subscription indexes.

Estimating the Memory Usage

The subscription indexes of the general Boolean algorithm comprise the minimum predicate count vector, the subscription location table, and the subscription trees. The two former structures contain one entry per registered subscription. Their memory requirements are thus not influenced by the application of pruning. The latter structure, on the other hand, represents the encoding of a subscription s_i itself. This encoding changes if subscription s_i is pruned to s_j.

We can use the size $mem(s_i)$ of the encoded subscription tree of subscription s_i as a measure for its memory requirements. In Section 5.4 (page 130), we described the memory requirements for subscription trees by $|p| \times (w(p)+1) + 2|op|$, with $|p|$ stating the number of predicates, and $|op|$ the number of Boolean operators, that is, inner nodes. Hence every pruning of s_i to s_j reduces the memory requirements for the encoding of the tree structure, as is our design goal.

Although the overall memory requirements of a pub-sub system when pruning are also influenced by predicate indexes, we chose to integrate the presented measure into BoP. The reason for this choice is the greater effect of subscription indexes on the change in memory requirements and the highly implementation-dependent influence of predicate indexes on this memory requirement.

Memory Improvement Measure

Based on our description of the memory requirements $mem(s_i)$ of a subscription s_i, we now define a memory-based ranking measure—the *memory improvement*. When pruning subscription s_i to s_j, the memory improvement is:

$$\Delta^{\approx}_{mem}(s_i, s_j) \quad = \quad mem(s_i) - mem(s_j).$$

This memory improvement $\Delta^{\approx}_{mem}(s_i, s_j)$ directly describes the difference (i.e., the reduction) in the encoding size of a subscription before and after pruning. Hence the larger the rank $\Delta^{\approx}_{mem}(s_i, s_j)$, the more reduction in memory requirements and thus the more beneficial the effect of pruning. Therefore, the best local optimization is the pruning operation that leads to the highest rank $\Delta^{\approx}_{mem}(s_i, s_j)$. For global decisions, pub-sub systems execute the determined local decisions in a descending order of the rank.

In contrast to the previous two measures (accuracy and efficiency), s_i in $\Delta^{\approx}_{mem}(s_i, s_j)$ directly refers to subscription s_i before being pruned to s_j. That is, s_i potentially describes an already pruned subscription. Using this policy allows the system to always incorporate the direct effect of pruning on the size of event routing tables. Conversely, using the unpruned subscription when performing several pruning operations for one subscription would consider a subsequent pruning as worthwhile if, for example, only the first operation results in a strong reduction in memory use. Our approach of treating each optimization decision individually, however, avoids this effect. A system thus optimizes based on the direct effect of each particular pruning operation.

The strongest reduction in memory use occurs when removing the largest possible branch of a subscription tree. For the memory-based variant, we thus additionally restrict the set of valid pruning operations as follows: the removal of a node n only constitutes a valid pruning operation if there exists no valid pruning option in the subtree that is rooted in n. Hence, systems only consider those pruning operations as valid that prune as near to the leaf nodes as possible.

For this ranking measure (as for the efficiency-based measure), it is also likely that various pruning options result in the same ranking value $\Delta^{\approx}_{mem}(s_i, s_j)$. Our implementation in BoP thus applies an extended policy to decide on the preferable pruning operation in such cases. We refer to Section 6.4.8 for details on this policy.

Validity of the Measure

Subscription indexes are not the only data structures that are subjected to a reduction in memory requirements when applying pruning. Predicate indexes, that is, predicate-subscription association table and one-dimensional indexes, also consume less memory due to optimization. The predicate-subscription association table always contains fewer associations whenever leaf nodes are removed from subscription trees. The effect on one-dimensional indexes depends on the commonality of the removed predicates. These structures are only reduced in size if all occurrences of a predicate are removed (which is rather unlikely, see Section 6.2.1).

Although we did not incorporate the alteration of the predicate-subscription association table into our ranking measure, $\Delta^{\approx}_{mem}(s_i, s_j)$ already includes the beneficial effect of removing predicates. Because the encoding of leaf nodes requires more memory than the encoding of inner nodes, the removal of predicates is weighted higher than the removal of Boolean operators. Our measure thus incorporates the stronger effects of pruning leaf nodes.

The additional restriction of valid pruning operations counteracts the aim of achieving the strongest possible reduction in memory requirements when pruning. In practice, however, this policy is not a real limitation: it just takes more pruning operations to remove the largest possible subtree of a subscription. In empirical experiments (see Chapter 8), this ranking measure still results in the largest reduction in memory whilst executing the least number of pruning operations.

Decisions for Example Subscription Class

Having presented the theory of the memory measure $\Delta^{\approx}_{mem}(s_i, s_j)$, we now give an example using a subscription of Class 3:

Example 6.5 (Memory improvement) *The memory requirements of the encoding of the original subscription s_i is $mem(s_i) = 7 \times 5 + 2 \times 6 = 47$ bytes.*

Different from Figure 6.3, the further restriction of valid pruning options for the memory-based measure excludes Pruning Option 2 (other valid operations exist in the pruned subtree) and thus leads to only five valid pruning operations. Performing these options, the memory requirements of the remaining subscriptions s_j are as follows (in bytes):

$$mem(s_j) = 5 \times 5 + 2 \times 4 = 33 \textit{ for Pruning Option 1},$$

$$mem(s_j) = 5 \times 5 + 2 \times 4 = 33 \textit{ for Pruning Option 3},$$
$$mem(s_j) = 6 \times 5 + 2 \times 4 = 38 \textit{ for Pruning Option 4},$$
$$mem(s_j) = 6 \times 5 + 2 \times 5 = 40 \textit{ for Pruning Option 5},$$
$$mem(s_j) = 6 \times 5 + 2 \times 5 = 40 \textit{ for Pruning Option 6}.$$

Applying the memory improvement measure, the ranking $\Delta^{\approx}_{mem}(s_i, s_j)$ *of these pruning operations are:*

$$\Delta^{\approx}_{mem}(s_i, s_j) = 47 - 33 = 14 \textit{ for Pruning Option 1},$$
$$\Delta^{\approx}_{mem}(s_i, s_j) = 47 - 33 = 14 \textit{ for Pruning Option 3},$$
$$\Delta^{\approx}_{mem}(s_i, s_j) = 47 - 38 = 9 \textit{ for Pruning Option 4},$$
$$\Delta^{\approx}_{mem}(s_i, s_j) = 47 - 40 = 7 \textit{ for Pruning Option 5},$$
$$\Delta^{\approx}_{mem}(s_i, s_j) = 47 - 40 = 7 \textit{ for Pruning Option 6}.$$

Based on these rankings, the system identifies Pruning Options 1 and 3 as preferable operations. As mentioned in Section 6.4.4, the extended policy of breaking such ties is then used to select between Options 1 and 3 as the best local pruning.

6.4.5 Pruning Based on Subscription Accuracy and Predicate Occurrence

In Section 6.4.2, we introduced an accuracy-based ranking measure for pruning operations. On the one hand, we argued that the incorporation of the overlapping relationships among subscriptions into this measure could improve its precision. On the other hand, the analysis of these relationships results in complex computation tasks, as in current routing approaches (see Section 2.5, page 47). Thus, within this section we take an indirect relation approach that incorporates the overlap among subscriptions based purely on information about the occurrence of their predicates, leading to an *accuracy and occurrence measure*.

The overall goal of the final degradation measure is to rank the removal of uncommon predicates higher than the removal of common predicates. The rank is additionally dependent on the increase in selectivity that is induced by a particular pruning operation. The motivation for this approach is that the preferred removal of uncommon predicates reduces the existing diversity among unpruned subscriptions. Hence it increases the similarity among pruned subscriptions. Increasing this similarity leads to a higher probability that subscriptions specify an interest in the same event messages. Hence the number of false positives due to pruning, and thus the increase in network load and additionally routed messages, is reduced.

This property of pruning decisions, in turn, is directly exploited by the filtering algorithm applied in BoP: the filtering shortcut, presented earlier, removes the need to apply the final subscription matching step for a large number of subscriptions. In particular when performing various pruning operations, and thus reducing subscriptions to very compact filter expressions, the benefit of this ranking measure increases. Additionally, the preferred removal of uncommon predicates reduces the memory requirements for predicate indexes and increases the efficiency of their evaluation. These effects are stronger than when removing arbitrary predicates.

Note, however, that this pruning measure is still applicable regardless of subscription commonality. The measure bases its pruning decisions on the occurrence of individual predicates only, and not on the commonality among whole subscriptions.

Estimating Predicate Occurrence

What is required by the pruning optimization is an efficient means of determining whether a pruned branch of a subscription removes highly common or rather uncommon predicates. To decide on this question, we apply a proportional measure to rank the degradation in predicate occurrence. This measure relates the predicate occurrence $occ(s_i)$ of a subscription s_i before pruning to its predicate occurrence $occ(s_j)$ after being pruned to s_j.

We define the *predicate occurrence of subscriptions* as follows:

Definition 6.2 (Predicate occurrence) *The predicate occurrence of a subscription $s \in \mathcal{S}$ describes the occurrences of its predicates by the numeric value $occ(s)$. This value is based on the set of registered subscriptions $\mathcal{S}$ and the predicates $\mathcal{P}(\mathcal{F})$ of the Boolean filter expression $\mathcal{F}$ of s. The calculation of $occ(s)$ works as follows:*

- *For a predicate p, $occ(p)$ equals the number of predicate-subscription associations (to be found in the predicate-subscription association table) for p.*

- *For a subscription s, it holds that:*

$$occ(s) \;=\; \sum_{p_i \in \mathcal{P}(\mathcal{F})} occ(p_i).$$

This notion of predicate occurrence fulfills our design goals of describing the usage of the predicates that are included within a (branch of a) subscription tree.

For leaf nodes, this information can be directly derived from the encoded predicate and predicate index structures, that is, the predicate occurrence equals the real usage of the predicate in the registered subscriptions. For inner nodes, both conjunctive and disjunctive branches, the predicate occurrence summarizes the individually derived predicate occurrences of child nodes.

Removing any branches of subscription trees results in a decrease of predicate occurrence according to the predicates in the pruned branch.

Selectivity and Occurrence Degradation Measure

Our ranking measure, the *estimated selectivity and occurrence degradation* $\Delta_{occ}^{\approx}(s_i, s_j)$, describes the proportional change in predicate occurrence combined with the absolute change in selectivity when performing pruning operations. It can thus be used to quantify the effect of pruning operations for both local and global optimization decisions.

We argued for the utilization of an absolute selectivity degradation measure in Section 6.4.2. However, for the predicate occurrence part of our combined degradation notion, we apply a proportional notion for the following reason: When considering the absolute change in predicate occurrence, pruning operations would be performed regardless of the predicate occurrence of the remaining (not pruned) parts of subscriptions. However, the remaining branches or, more generally, the subscriptions before pruning, influence the effect of a fixed reduction in predicate occurrence. For example, a reduction in predicate occurrence by value v should be regarded as more valuable if the overall, unpruned subscription has a predicate occurrence of $occ(s_i) = 100 \times v$ than of $occ(s_j) = 5 \times v$. The reason for this preference is that subscriptions that involve both common and uncommon predicates (e.g., s_i) should be pruned before those subscriptions that only include uncommon predicates (e.g., s_j). If there are no common predicates, a pruning can never lead to the desired effect of keeping the common ones.

We define the estimated selectivity and occurrence degradation of a subscription s_i that is pruned to s_j as follows (it extends the estimated selectivity degradation, see Section 6.4.2):

$$\Delta^{\approx}_{occ}(s_i, s_j) \quad = \quad \frac{occ(s_i)}{occ(s_j)} \times \Delta^{\approx}_{sel}(s_k, s_j).$$

For the selectivity part of $\Delta^{\approx}_{occ}(s_i, s_j)$, s_k refers to the original, unpruned subscription. However, for the occurrence part, BOP uses the value $occ(s_i)$ of subscription s_i before the current pruning operation in its calculations. This handling is required to express the change in predicate occurrence of the currently performed pruning operation, that is, of the currently removed branch. The occurrence part of the degradation, $\frac{occ(s_i)}{occ(s_j)}$, would otherwise state the total change for all pruning that is performed. It would, in fact, increase as more pruning operations are executed and thus not adhere to its objective of benefiting the removal of uncommon predicates (the removal would be counteracted when repeatedly pruning subscriptions).

Validity of the Measure

Our accuracy and predicate occurrence ranking measure estimates the influence of pruning operations based on the introduced predicate occurrence measure. Hence, the derived ranking $\Delta^{\approx}_{occ}(s_i, s_j)$ can only incorporate those effects that are captured by the underlying predicate occurrence concept $occ(s)$. The overall goal of the occurrence-based pruning variant is to reduce the number of false positives. These false positives depend on more than the usage of individual predicates, as captured by $occ(s)$:

Even if the removal of uncommon predicates is preferred when pruning, the remaining subscriptions might not describe similar interests. This is because the combination of these predicates might still lead to diverse specifications in subscriptions. On the other hand, an uncommon predicate p might be covered by very common ones. The removal of p thus does not decrease the diversity of subscriptions because a part of the pruned subscription was already covered before optimizing.

Obviously, the introduced measure $\Delta^{\approx}_{occ}(s_i, s_j)$ shares the same validity issues as $\Delta^{\approx}_{sel}(s_i, s_j)$, as analyzed in Section 6.4.2. Additionally, the straightforward multiplicative combination of the selectivity and occurrence components in $\Delta^{\approx}_{occ}(s_i, s_j)$ is only one alternative to merging these two constituents of the ranking measure.

Despite this, the introduced ranking measure led to beneficial results in empirical experiments (see Chapter 8).

Decisions for Example Subscription Class

To exemplify the calculation of the estimated selectivity and occurrence degradation measure, we have to make assumptions about the usage of predicates. We do so in the following examples, again using a subscription of Subscription Class 3. We start with determining the predicate occurrence of subscriptions:

Example 6.6 (Predicate occurrence) *Subscriptions of Class 3 (see Figure 3.3) contain seven predicates, p_1 to p_7. Let us assume the following predicate occurrences $occ(p_1)$ to $occ(p_7)$ in the following (the orders of magnitude are insignificant because these predicate occurrences are incorporated proportionally later on):*

$$occ(p_1) = 100, occ(p_2) = 30, occ(p_3) = 500, occ(p_4) = 70,$$
$$occ(p_5) = occ(p_7) = 60, occ(p_6) = 400.$$

Based on these assumptions, for the predicate occurrence of the original subscription s_i it holds that:

$$occ(s_i) = \sum_{k=1...7} occ(p_k) = 100 + 30 + 500 + 70 + 60 + 400 + 60 = 1,220.$$

For the pruned subscriptions s_j (see Figure 6.3 for these pruning operations), it holds that (ordered by option number):

$$\begin{aligned}
occ(s_j) = \sum_{k=3...7} occ(p_k) &= 500 + 70 + 60 + 400 + 60 = 1,090, \\
occ(s_j) = \sum_{k=1...2} occ(p_k) &= 100 + 30 = 130, \\
occ(s_j) = \sum_{k=1...5} occ(p_k) &= 100 + 30 + 500 + 70 + 60 = 760, \\
occ(s_j) = \sum_{k=1,2,4...7} occ(p_k) &= 100 + 30 + 70 + 60 + 400 + 60 = 720, \\
occ(s_j) = \sum_{k=1...3,5...7} occ(p_k) &= 100 + 30 + 500 + 60 + 400 + 60 = 1,150,
\end{aligned}$$

$$occ(s_j) = \sum_{k=1\ldots4,6,7} occ(p_k) \quad = \quad 100 + 30 + 500 + 70 + 400 + 60 = 1,160.$$

Based on these predicate occurrences, we now provide an example of the calculation of the degradation measure $\Delta^{\approx}_{occ}(s_i, s_j)$:

Example 6.7 (Selectivity and predicate occurrence degradation) *The selectivity and occurrence degradation $\Delta^{\approx}_{occ}(s_i, s_j)$ consists of a selectivity and a predicate occurrence part. We already calculated the selectivity part of this measure in Example 6.3 (page 171). For the occurrence part, we can use the calculations from the previous example. Based on this information, it holds that (ordered by the pruning option number):*

$$\begin{aligned}
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{1,090} \times 0.154 \approx 0.172,\\
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{130} \times 0.015 \approx 0.141,\\
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{760} \times 0.00034 \approx 0.00055,\\
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{720} \times 0.01239 \approx 0.021,\\
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{1,150} \times 0.00019 \approx 0.000202,\\
\Delta^{\approx}_{occ}(s_i, s_j) &= \frac{1,220}{1,160} \times 0.00049 \approx 0.000515.
\end{aligned}$$

Hence the order of pruning operations is Option 5, 6, 3, 4, 2, and 1. Comparing this result with the pure selectivity degradation measure (see Example 6.3), one realizes a change in the ranking of Pruning Options 6 and 3. Whereas the pure selectivity-based measure favors Option 3 over Option 6, the new combined degradation chooses Option 6 before Option 3. The reason for this change is that predicates p_6 and p_7 have a higher predicate occurrence than p_5. The occurrence part of the combined measure thus favors Option 6 over Option 3 (smaller occurrence part), which leads to a smaller, that is, preferred, ranking for Option 6.

6.4.6 Pruning Based on Subscription Accuracy and Distance

In the previous section, we introduced one valuable extension to our accuracy-based ranking measure. Additionally, we identified another attribute, the distance from the subscriber, that could advantageously extend this measure, leading to a novel ranking measure: the *accuracy and distance measure*.

When utilizing this measure, the applied event routing algorithm requires a minor extension: Brokers need to know about the *distance* to the subscriber of a subscription s_i, referred to as *subscriber distance*, $dist(s_i)$, in the following. The simplest representation of this subscriber distance is the number of hops from a broker to the local broker $B(s_i)$ of s_i. This distance can be straightforwardly distributed to brokers when subscriptions are registered (increasing a hop count per forwarding); it is thus known to all brokers in the network.

Knowing the subscriber distance allows the pub-sub system to exploit this attribute when optimizing, that is, when pruning subscriptions. The objective of the accuracy and distance measure is to additionally (next to the accuracy degradation, see Section 6.4.2) weight pruning options according to the subscriber distance. The pruning of a subscription s_i is preferred if it shows a large subscriber distance $dist(s_i)$.

The reason for this ranking is found in the distribution of false positives in the network: Preferring the pruning of far distant subscriptions decreases the probability of false positives reaching the local broker. In fact, false positives are likely to be filtered out on their way to the local broker of a subscription. This is the case because a low selectivity degradation of a subscription (only estimated and thus potentially incorrect) is weighted high if the optimization is performed near to the subscriber. Hence, if a well-ranked pruning option (or, in fact, any pruning) leads to various false positives, these false positives are not distributed along the whole path to the subscriber. Instead the ranking measure prevents this distribution in the network due to its property of considering the locality of pruning decisions.

So far we have not integrated the accuracy and distance measure into BoP. The main reason for this is the required extension of the applied routing algorithm, which opposes our goal of utilizing existing protocols. However, we plan to integrate the measure into BoP in the future, and to evaluate and compare its usefulness. This future work also includes the consideration of broader definitions of distance, such as the available bandwidth on the path to the local broker, or other parameters of the involved machines and connections.

6.4.7 Pruning Based on Combined Parameters

In the previous sections, we presented five measures that rank pruning operations according to different target parameters. The latter two of these measures were extensions to the accuracy-based measure. They combine the accuracy parameter (estimating the increase in network load) with two subscription properties—the predicate occurrence of subscriptions and the subscriber distance. Obviously, there are several ways of weighting the influence of these different factors in the resulting ranking measure.

Most of the introduced target parameters are independent of each other; they also cannot be transformed into each other; and they might even be conflicting. Therefore, the simultaneous consideration of all parameters leads to a multi-criteria optimization problem [Ste86]. Tackling pruning decisions in this way allows tailoring of the optimization to the current application scenario and, potentially, leads to an advanced overall optimization effect. Within this dissertation, we do not go into detail on such a multi-criteria optimization approach but leave the research to future work. However, within BoP we do consider several target parameters simultaneously if there exists a set of optimal solutions for the currently chosen parameter. We elaborate on this advanced handling of pruning operations in the following subsection.

6.4.8 Pruning In Case of Ties

For the four ranking measures presented in Sections 6.4.2 to 6.4.5, different pruning options (for the actual target parameter) might show the same ranking value $\Delta(s_i, s_j)$. The likelihood of these ties varies for the different measures. Ties can either occur for pruning options of one subscription (local decisions) or for pruning options of several registered subscriptions (global decisions among preferred local decisions). In both cases, a pub-sub system can refer to the ranking value of a parameter other than the currently applied one to reach its final optimization decision.

The employed order of parameters depends on the requirements of the actual application scenario, and the registered subscriptions and published event messages. Within BoP, and thus in our empirical experiments (Chapter 8), we apply the following orders of parameters when pruning subscriptions s_i to s_j:

- $\Delta^{\approx}_{sel}(s_i, s_j)$, $\Delta^{\approx}_{eff}(s_i, s_j)$, $\Delta^{\approx}_{mem}(s_i, s_j)$ for accuracy-based pruning
- $\Delta^{\approx}_{eff}(s_i, s_j)$, $\Delta^{\approx}_{sel}(s_i, s_j)$, $\Delta^{\approx}_{mem}(s_i, s_j)$ for efficiency-based pruning
- $\Delta^{\approx}_{mem}(s_i, s_j)$, $\Delta^{\approx}_{sel}(s_i, s_j)$, $\Delta^{\approx}_{eff}(s_i, s_j)$ for memory-based pruning

- $\Delta^{\approx}_{occ}(s_i, s_j)$, $\Delta^{\approx}_{sel}(s_i, s_j)$, $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$ for accuracy and occurrence-based pruning

If all three of these chosen ranking measures show the same value, the preferred pruning option is arbitrary. We illustrate this extended strategy to select the preferred pruning option in the following example:

Example 6.8 (Breaking a Tie) *In Example 6.4 (page 176), we presented the ranking of pruning operations when using the efficiency improvement measure. Using only this parameter for pruning decisions results in a tie among five pruning options. However, applying the extended strategy, which is presented in this section, leads to a definite order of these pruning operations. This decision can be found after referring to the ranking of two of the three parameters ($\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$ and $\Delta^{\approx}_{sel}(s_i, s_j)$). We already calculated the rankings for these measures in Example 6.3 (page 171) and Example 6.4 (page 176) and use our results in the following:*

$\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$ (Example 6.4) equally ranks Pruning Options 1, 3, 4, 5, and 6. However, $\Delta^{\approx}_{sel}(s_i, s_j)$ (Example 6.3) leads to different rankings for these five options, resulting in the final order: the preferred pruning is Option 5 followed by Options 3, 6, 4, and 1. Option 2, having a greater ranking value for $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$, is the least preferred pruning option.

6.5 Variants of Subscription Pruning

Having introduced the general idea of subscription pruning and a wide range of ranking measures to determine an order among all possible pruning operations, we now elaborate on different variants of this optimization. We identified two main variants of the application of pruning—post-pruning (Section 6.5.1) and pre-pruning (Section 6.5.2). As well as individually applying these variants, they can be used collectively, as presented in Section 6.5.3.

6.5.1 Post-pruning

In the *post-pruning* variant, brokers individually perform subscription pruning to achieve an optimization of the system. In combination with the applied routing algorithm, broker components forward each incoming subscription s in its original, that is, unpruned, form. This incoming subscription s is either integrated as is (if s is a local subscription), or potentially pruned (if s is a

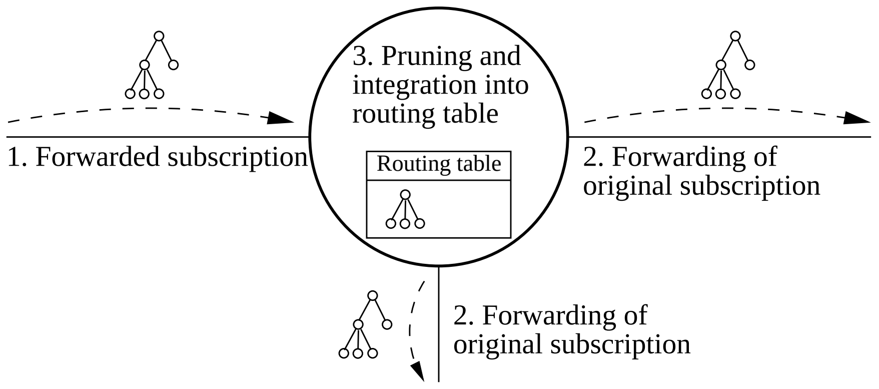

3. Pruning and integration into routing table
1. Forwarded subscription
Routing table
2. Forwarding of original subscription
2. Forwarding of original subscription

Figure 6.6: Post-pruning in one broker (center of the figure): the incoming subscription is forwarded to neighbors and a (potentially) pruned version of this subscription is inserted into the local event routing table.

non-local subscription), as described in Section 6.6. This policy of handling subscriptions provides the name to this pruning variant: subscriptions are only pruned post being forwarded.

Proceeding in that way allows brokers to independently perform pruning based on their current situation. Global pruning decisions in brokers are thus reached from the viewpoint of individual system components. The pub-sub system should apply a global pruning policy, for example, to co-ordinate the number of pruning operations each broker performs. Without such a policy, every component only considers its individual routing load when optimizing. Potentially, this could lead to an overload in neighbor brokers, for example, if routing entries become highly inaccurate and thus large numbers of false positives are forwarded to neighbors.

We give an overview of the post-pruning variant in Figure 6.6: unpruned subscriptions are forwarded to the two neighbor brokers whereas (independently) pruned subscriptions are integrated into event routing tables. These neighbors, in turn, independently perform pruning decisions but forward the original, unpruned subscriptions.

When applying post-pruning, pruning operations can be performed before subscriptions are inserted into routing tables, for example, up to a certain threshold of the (global) value of the employed ranking measure. Alternatively, subscriptions, that is, routing entries, can be pruned whenever a broker decides to optimize the system, also up to a specific threshold of the utilized ranking measure.

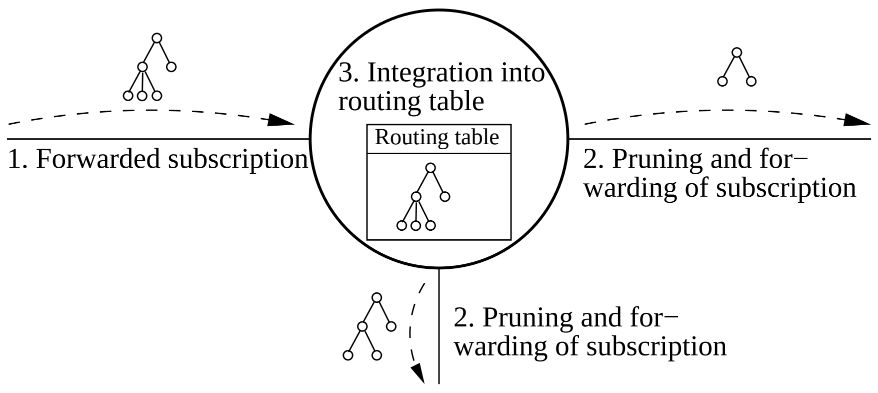

3. Integration into routing table
Routing table
1. Forwarded subscription
2. Pruning and for−warding of subscription
2. Pruning and for−warding of subscription

Figure 6.7: Pre-pruning in one broker (center of the figure): (potentially) pruned subscriptions are forwarded to neighbors and this version of the incoming subscription is inserted into the local event routing table.

6.5.2 Pre-pruning

In the other pruning option, *pre-pruning*, broker components prune subscriptions before they are forwarded to neighbors in the network. Subscriptions might thus reach neighbor brokers in an already altered, that is, pruned, form. Using this pure pre-pruning variant, brokers always integrate a subscription s in their routing tables in the way s was forwarded by a neighbor.

Applying pre-pruning provides pub-sub systems with the possibility to optimize the routing in the network as a whole. Next to selecting the preferred pruning options based on the previously presented ranking measures, brokers can apply different policies for different neighbors in the pub-sub system. These policies can be based on heuristics or statistical information from neighbors, for example, their memory usage or the available bandwidth of the respective network connections. In particular, in heterogeneous networks involving variably equipped machines, this option is preferable with respect to optimizing the overall efficiency of the distributed pub-sub system.

An implication of pre-pruning is that pruning decisions are potentially based on non-local information. That is, subscriptions that are integrated in the event routing tables of broker B are pruned in a component other than B. Taking the accuracy-based ranking measure as an example, the selectivity estimation could become inaccurate if the distribution of event messages that is sent by publishers is not relatively evenly distributed among brokers.

In Figure 6.7, we illustrate an example of pre-pruning: a forwarded subscription s is integrated into the local routing table. For the two neighbors, however, s is pruned in different ways and forwarded in this altered form.

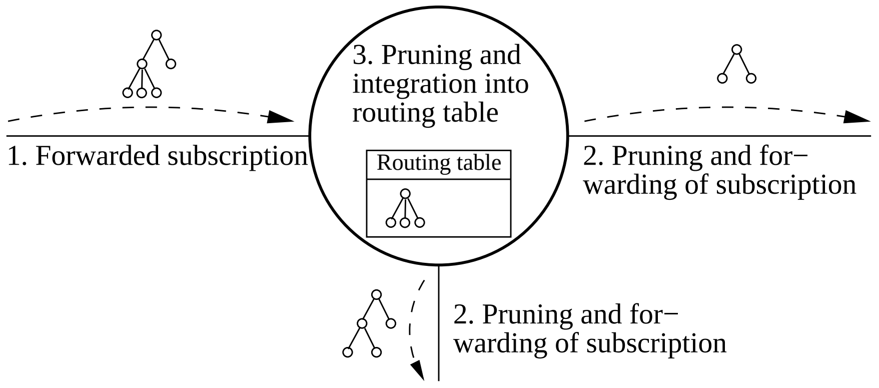

3. Pruning and integration into routing table
Routing table
1. Forwarded subscription
2. Pruning and for− warding of subscription
2. Pruning and for− warding of subscription

Figure 6.8: Combined pruning in one broker (center of the figure): (potentially) pruned subscriptions are forwarded to neighbors and a (potentially) pruned version of the incoming subscription is inserted into the local event routing table.

These neighbors then integrate the pruned variant into their routing tables and further decide on the (potential) pruning before forwarding. Pruning operations in the pre-pruning variant always need to be performed before the actual forwarding process. Pruning can either be executed up to a global threshold or, as mentioned previously, on a per-broker basis. Obviously, these thresholds might adapt to the current system status.

6.5.3 Combined pruning

Finally, there exists a hybrid of the two previously introduced subscription pruning variants, *combined pruning*. In this case, subscriptions might reach broker components in an already pruned form. Brokers then decide on both the pruning of their own event routing entries, and the pruning for neighbors, before further forwarding an incoming subscription.

We illustrate an (extreme) example of this approach in Figure 6.8: the incoming subscription is firstly pruned for the own event routing table, and secondly pruned in different ways for the two neighbors in the network.

Having presented these three pruning variants, we give details about the practical implementation of subscription pruning in the following section.

6.6 Practical Subscription Pruning

Applying the subscription pruning optimization in pub-sub systems easily integrates into the existing filtering and routing structures, without requiring internal modifications. What is needed to practically support subscription pruning is simply an extension of the existing system, as presented in the following subsections.

6.6.1 Pruning Structures

Regardless of the applied pruning variant, the pub-sub system requires the means to calculate the applied ranking measure to reach its optimization decisions. BoP currently supports the four measures that were presented in Sections 6.4.2 to 6.4.5. According to the employed measure, the required calculation information can either be derived from existing filtering structures or needs to be obtained additionally.

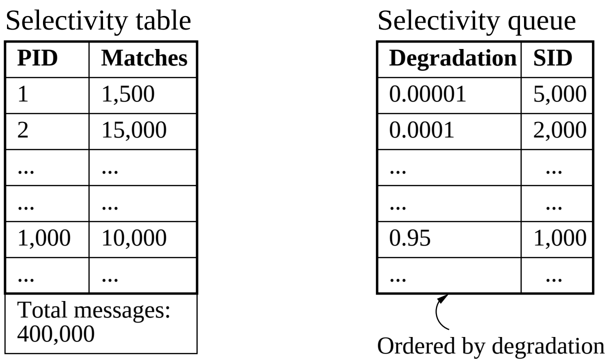
Accuracy-based Ranking Measure

The accuracy-based ranking measure utilizes the selectivity of predicates to reach optimization decisions. Because neither the original filtering algorithm nor the original routing algorithm requires this information, BoP includes a *selectivity table* (see Figure 6.9, left) that administers the selectivities of predicates. This table maps predicate identifiers p to a counter $|msg_f(p)|$ that is increased for each fulfilled predicate of incoming event messages.

Based on this information (as well as the total number of filtered messages $|msg_t|$), BoP can firstly estimate the selectivity $sel^{\approx}(s_i)$ of any subscription s_i and secondly derive the selectivity degradation $\Delta^{\approx}_{sel}(s_i, s_j)$ for any pruning of s_i to s_j. Thirdly, having obtained these selectivities allows for the application of the efficiency-enhancing ordering extension of the filtering algorithm (see Section 4.5.3, page 113).

Efficiency-based Ranking Measure

The efficiency-based ranking measure is built on the minimal number of fulfilled predicates. The applied filtering algorithm already utilizes this information, which is stored in the minimum predicate count vector for each registered subscription s_i. For pruned subscriptions s_j, BoP calculates $p_{min}(s_j)$, and thus does not need additional data structures for this pruning variant which is based on the efficiency improvement $\Delta^{\approx}_{\mathit{eff}}(s_i, s_j)$.



Selectivity table

PID	Matches
1	1,500
2	15,000
...	...
...	...
1,000	10,000
...	...
Total messages: 400,000	

Selectivity queue

Degradation	SID
0.00001	5,000
0.0001	2,000
...	...
...	...
0.95	1,000
...	...

Ordered by degradation

Figure 6.9: Overview of additional pruning structures: selectivity table (left) and selectivity queue (right).

Memory-based Ranking Measure

The application of the memory-based ranking measure also does not necessitate novel data structures. The size $mem(s_i)$ of the subscription trees of registered subscriptions s_i is directly given by their encoding that is used by the filtering algorithm. The sizes of pruned subscriptions s_j can also be straightforwardly determined by BoP by applying the utilized encoding scheme. Hence the memory improvement $\Delta^{\approx}_{mem}(s_i, s_j)$ is known to the system.

Accuracy and Occurrence-based Ranking Measure

The accuracy and occurrence-based ranking measure consists of two components: for the accuracy component, BoP requires the same addition as for the pure accuracy-based ranking measure, a selectivity table. The predicate occurrence component, however, is completely derived from the occurrences of predicates. This information is already required in the filtering algorithm and is stored in the predicate-subscription association table.

Therefore, BoP can determine the selectivity and predicate occurrence degradation $\Delta^{\approx}_{occ}(s_i, s_j)$ for any registered subscription s_i that is pruned to s_j. It can thus appropriately rank pruning operations.

6.6.2 Bulk Pruning

Post-pruning offers the opportunity to perform *bulk pruning*, that is, at any point in time the pub-sub system might decide to optimize its routing tables and apply subscription pruning (see Section 6.5.1). For this purpose, BoP utilizes a *degradation queue* (see Figure 6.9, right) for accuracy, and accuracy and occurrence-based pruning; an *improvement queue* is applied for efficiency and memory-based pruning. In the following, we elaborate on the degradation queue in combination with accuracy-based pruning. The improvement queue works analogously (except for the order of its entries).

A degradation queue implements a priority queue (see, e.g., [CLRS01]), storing tuples that consist of a degradation value $\Delta(s_i, s_j)$ and a subscription identifier s_i. These elements are sorted by the queue in an ascending order of their degradation value components ($\Delta(s_i, s_j)$). Hence the tuple that specifies the least degradation value (i.e., that involves the best ranking) is stored on top of the queue and can be accessed efficiently.

To allow for the application of bulk pruning, BoP calculates the preferred pruning option for each subscription s_i (leading to s_j) at its point of registration. The system then creates a tuple $(\Delta(s_i, s_j), s_i)$ and inserts it into the degradation queue. When using Fibonacci heaps [FT87], this insertion works in amortized constant time and thus only creates an insignificant overhead.

Most importantly, when proceeding in that way, at any point in time the degradation queue allows for the efficient access of the pruning option that leads to the least degradation, that is, that involves the best ranking value. Bulk pruning involves the following steps:

1. Remove the top element $(\Delta(s_i, s_j), s_i)$ from the degradation queue.

2. Perform[3] the preferred pruning of subscription s_i, which is leading to s_j.

3. Remove subscription s_i from index structures and index subscription s_j.

4. Insert tuple $(\Delta(s_j, s_k), s_j)$ into the degradation queue, whereas s_k states the preferable pruning of subscription s_j.

Bulk pruning is executed until the desired amount of optimization is reached.

Incorporating Changes in Rankings

For both accuracy-based pruning variants BoP might experience changes in the calculated ranking measures once they are inserted into the degradation queue. Such changes, for example, occur if the distributions of incoming event messages vary significantly over time. In the following, we present three strategies to cope with such potential changes. In the efficiency and memory-based

[3]We refer to the following paragraph for a refinement of this step.

pruning variants, on the other hand, the calculated rankings can never be altered once they are inserted (both $p_{min}(s_i)$ and $mem(s_i)$ merely depend on subscription s_i, and neither on other subscriptions nor filtered event messages).

Strict Execution. This strategy ignores any changes in previously calculated degradation values. It thus always performs the pruning option that is described by the top element in the degradation queue. An obvious consequence of this strategy is the execution of pruning operations that show a non-minimal ranking value at the time of optimizing. Hence the system does not always perform pruning in order of the applied measure.

Statically Restricted Execution. The second strategy allows changes up to a certain threshold in the calculated ranking between the time of insertion into the degradation queue and the actual execution of a pruning. These permitted changes are based only on the stored and the newly calculated ranking, not on the rankings of any other registered subscriptions. Two straightforward options for thresholds are to allow fixed differences between stored and new rankings, or to allow a proportional difference between these two values, for example, up to a certain percentage.

Whenever the change in the ranking value exceeds the permitted range, the system neglects this pruning option. It then re-inserts an element with the newly calculated ranking into the degradation queue and again removes the (now new) top element from the queue (cf. Step 1 of bulk pruning).

Dynamically Restricted Execution. The final execution strategy relates the new ranking value of the top element (provided it changed) to the ranking value of the element below the top. The pruning is either only executed if the new ranking is still less than the ranking of the element below the top, or if it differs up to a permitted threshold (fixed or proportional). If the pruning is rejected due to its difference, a new element is inserted into the queue (stating the updated situation) and the (now new) top element is considered, as presented before.

Further Considerations. In practice, one should restrict the number of re-insertions into the queue in case of rejected pruning operations. In particular, in case of only slightly differing ranking values, the number of re-insertion cycles might otherwise become high, leading to inefficient pruning decisions.

Another important point to note is that not only the ranking of the top element of the queue might change over time. In fact, all subscriptions are subject to change in distributions of event messages. Thus executing pruning operations in their exact ranking order does require creation of the degradation queue at the time of optimizing and not at the time of subscription registration. If the event load in the system allows for this, the optimization results comply more with the utilized ranking measure.

However, it might become superfluous to consider changes in rankings because of the property of bulk pruning to execute various pruning operations in batch. This is due to the fact that the actual order of pruning does not alter the overall optimization result. For example, if all registered subscriptions are pruned once according to their original rankings, a change in the pruning order of these subscriptions still leads to the same overall optimization result. We integrated this strict execution measure into BoP.

6.6.3 Deregistrations

Routing optimizations other than subscription pruning (covering, merging, and summarization) show the drawback of a strong network and processing overhead when deregistering subscriptions. We elaborated on this disadvantageous behavior in detail in Section 2.5.

For all of the introduced pruning options, on the other hand, the deregistration process works as in the case of un-optimized routing. That is, the deregistration is forwarded to neighbor brokers in the network. These brokers then remove the respective subscription from their index structures. If the non-local subscription to be deregistered was pruned, its pruned version is removed. The only requirement for the support of deregistrations is the existence of unique subscription identifiers, as is our assumption.

This advantageous behavior of subscription pruning compared to other optimizations weakens a potential argument against subscription pruning of distributing all subscriptions that are registered. Firstly, all other optimizations except covering also need to distribute all subscriptions. Covering might only avoid this effect if its strong assumptions are met. Secondly, the system load for deregistrations when applying any recent optimization is higher than in the un-optimized system. Summing up, the overall cost when using existing optimizations is even higher than in the case of subscription pruning.

Due to the reduction of the complexity of routing entries when pruning, subscription pruning supports deregistrations more efficiently than the unoptimized system. This is because the applied index structures are reduced in their sizes and numbers of entries.

6.7 Related Work

Having introduced the details of our routing optimization for general Boolean subscriptions, we relate this novel approach to current works in Section 6.7.1. Section 6.7.2 then gives details about existing approaches to estimate the selectivity of Boolean queries.

6.7.1 Event Routing Optimizations

We gave an overview of existing event routing optimizations in Section 2.5: subscription covering [CRW01, CF03, LHJ05, MF01, OJPA06], subscription merging [CBGM03, LHJ05, MF01], and subscription summarization [TE02, TE04, WQV$^+$04]. We here refer to the respective section for a complete evaluation of these approaches and only repeat our main findings for the sake of completeness:

Our evaluation included an analysis of these three optimizations with respect to the parameters optimization applicability, support of deregistrations, internal subscription model, and memory usage. All recent routing optimizations show a range of shortcomings with respect to these evaluated parameters (only applicable if assumptions are met, large overhead in case of deregistrations, only conjunctive subscriptions are efficiently supported, and a reduction in memory usage only exists if optimization assumptions are met).

These limitations of recent approaches led to our five design goals for routing optimizations that were presented in Section 6.1. Our theoretical analyses of both predicate replacement and subscription pruning (see Section 6.2.2 and Section 6.3.3) showed that both proposals largely comply with these goals. Subscription pruning has a higher optimization potential than predicate replacement. We show the results of an empirical evaluation of the promising pruning approach in Chapter 8.

Both subscription pruning and predicate replacement follow different optimization principles than recent approaches. We elaborated on these differences in detail in Section 6.1. The crucial point in this respect is the active manipulation of event routing entries (i.e., subscriptions) by our novel optimizations.

Recent approaches, on the other hand, try to exploit existing relationships among subscriptions to achieve their optimization. Looking at subscription pruning and predicate replacement in terms of current work, one realizes that our novel approaches actively create covering subscriptions in non-local brokers instead of passively depending on existing covering relationships. Additionally, subscription pruning can be used to solve the imperfect merging task. It can also be jointly applied in conjunction with recent optimizations due to their opposing strategies.

In [EFGH02], Eugster and colleagues briefly sketch the broad approach of filter weakening. Filter weakening could be seen as a predecessor of our approach of generalizing subscriptions. However, also the weakening approach in [EFGH02] is restricted to conjunctive subscriptions (as all other current solutions). There is no work on how to broaden subscriptions in practice, except the vague idea of basing it on their generality [EFGH02].

6.7.2 Selectivity Estimations

Estimating the selectivity of queries is researched in the context of database management systems, for example, in [CKKM00, PI97]. However, such approaches either require conjunctive queries, or the conversion of queries into disjunctive or conjunctive normal forms.

Apart from the time complexity that is required for these selectivity estimations and the memory consumption of the involved data structures, canonical conversion leads to an exponential space complexity (see Section 2.6, page 58). The memory consumption, however, is crucial in context of content-based pub-sub systems, due to their sole application of main memory filtering and routing algorithms (see Section 2.2, page 23). Current selectivity estimation solutions [CKKM00, PI97], however, are applicable to database management systems, which can convert queries to canonical forms (see Section 2.1.1, page 15) and whose problem definition is opposite to the problem definition of pub-sub systems.

6.8 Summary

In this chapter, we introduced the first event routing optimizations that are practically applicable to general Boolean subscriptions. Subscription pruning was identified as the more promising of the two presented approaches because it strongly optimizes pub-sub systems with respect to their quality measures.

The proposed subscription pruning optimization seamlessly integrates with the applied filtering and routing algorithms, and can be easily tailored to a range of optimization parameters. Altogether we introduced optimization strategies for six different target dimensions. Our pub-sub prototype BoP currently supports four of these strategies, customizing subscription pruning to optimize the system either according to the increase in network load or efficiency, or according to an effective decrease in memory usage. Subscription pruning is thus flexible enough to be applied to the improvement of different quality measures for pub-sub systems.

By introducing subscription pruning, we provided the second and final step in the support of general Boolean subscriptions in content-based pub-sub systems. To fully integrate the general Boolean pub-sub model (including advertisements) into these systems, we still need to provide for the handling of general Boolean advertisements. We take this concluding step in the following chapter.

Chapter 7

Supporting General Boolean Advertisements

In this chapter, we provide the final milestone to support the general Boolean pub-sub model: we allow for general Boolean advertisements in content-based pub-sub systems. The advantages of more general (than conjunctive) advertisements with respect to efficiency and scalability are similar to those of general Boolean subscriptions. Additionally, general Boolean advertisements allow publishers to more precisely describe their future event messages (see Chapter 3 for examples). This higher precision inherently leads to the forwarding of fewer subscriptions in the network and thus to smaller subscription routing tables.

To support general Boolean advertisements, BoP needs to solve two tasks in content-based pub-sub systems. We firstly propose a method to calculate the overlapping relationships between general Boolean subscriptions and advertisements. Secondly, we introduce advertisement pruning, the first designated advertisement-based routing optimization for pub-sub systems. Advertisement pruning is specifically tailored to optimize the applied subscription routing tables and will eventually allow us to prove Part 2b of our central hypothesis (page 6).

We structure this chapter as follows: In Section 7.1, we introduce the semantics and definition of general Boolean advertisements in BoP. These definitions are fundamental for the algorithm to calculate the overlap between subscriptions and advertisements, which is presented in Section 7.2. Advertisement pruning, our advertisement-based optimization, is then described in Section 7.3. Finally, we investigate related work in Section 7.4.

7.1 Advertisements: Semantics and Definition

In Section 4.1 (page 96), we provided precise definitions of event messages and general Boolean subscriptions in BoP. In this section, we expand these descriptions to the remaining concept of advertisements, extending their preliminary introduction in Section 2.1.1 (page 13).

Advertisements (as subscriptions and event messages) also utilize the notion of event types, as introduced in Section 4.1.1 (page 96). We define general Boolean advertisements as follows. This definition is similar to that of subscriptions; however, the semantics of advertisements is greatly dissimilar (cf. Section 4.1.2).

Definition 7.1 (Advertisement) *An* advertisement *a is a tuple specifying an event type name T^n and a Boolean filter expression $\mathcal{F}$, $a = (T^n, \mathcal{F})$, with $\mathcal{F}$ being a Boolean combination of predicates using the operators conjunction, disjunction, and negation. The set of predicates used in $\mathcal{F}$ is denoted by $\mathcal{P}(\mathcal{F})$. Each predicate $p \in \mathcal{P}(\mathcal{F})$ is an attribute-function-operand triple (i.e., a triple containing an attribute name, a filter function, and an operand): $p = (a^n, f, op)$.*

Each predicate p_i of the Boolean filter expression $\mathcal{F}$ of advertisement a has to specify one of the attribute names (e.g., a^n_j) that belong to the event type of a (i.e., $a^n_j \in T$), a filter function (e.g., f_i) that is included in the set of functions (e.g., a^f_j) specified by this attribute a^s_j, and an operand being valid as second variable for this filter function (i.e., f_i). That is, given an advertisement $a = (T^n, \mathcal{F})$, it holds:

$$\forall (a^n_i, f_i, op_i) \in \mathcal{P}(\mathcal{F}) : \exists a^s_j \in T : a^n_j = a^n_i \wedge f_i \in a^f_j \wedge op_i \in f^{op}_i.$$

This symmetrical definition of subscriptions and advertisements leads to the advantageous property of the calculation equivalence of the advertisement-subscription and the subscription-advertisement overlapping relationship (see Section 2.1.3, page 20). That is, one can apply the same algorithm for both directions of computation (see Section 7.2). Having defined advertisements in this way, we refine our notion of *conforming event messages* as follows (from Section 2.1.1):

Definition 7.2 (Conforming event message) *A message $e_j = (T^n_j, \mathcal{A}_j)$* conforms *to an advertisement $a_i = (T^n_i, \mathcal{F}_i)$ if, and only if:*

1. *Advertisement a_i and event e_j specify the same event type,that is, $T_i^n = T_j^n$.*

2. *The Boolean filter expression $\mathcal{F}_i$ evaluates to* true *on event e_j. For this evaluation, each predicate $p_i = (a_i^n, f_i, op_i)$ with $p_i \in \mathcal{P}(\mathcal{F}_i)$ gets assigned the result of the function $f_i(a_l^v, op_i)$ with $a_l^s \in \mathcal{A}_j \wedge a_l^n = a_i^n$. Then, the Boolean combination of these results, stated by $\mathcal{F}_i$, is evaluated.*

Symmetrically to subscriptions, our Boolean model does not require advertisements to specify predicates for all attribute specifications of its event type. The semantics in this case is that the publisher of such an advertisement, potentially, sends messages that involve all values for this attribute. This property also holds if the structure of the filter expression of an advertisement combines predicates in a way that all predicates referring to a particular attribute specification might evaluate to *false* in order to lead to a *true* filter expression.

If several predicates in the filter expression of an advertisement refer to the same attribute specification, the semantics depends on the Boolean operator that is used for their combination, for example, the disjunction of two predicates p_i and p_j describes that for every conforming message e_k either p_i or p_j (or both) evaluate to *true*[1].

In Section 3.4, we defined a set of example advertisement classes (as well as particular advertisements) in our online auction application scenario. They are graphically illustrated in Figures 3.4 to 3.11 by what was introduced as advertisement trees. These tree structures are used for both graphical and internal representation purposes. They follow the same definition as subscription trees. We thus refrain from the detailed definition of advertisement trees here but refer to the respective section (Section 4.1.2, page 100) on subscription trees.

7.2 Calculating the Overlapping Relationship

The first step in supporting general Boolean advertisements regards the calculation of the overlapping relationship. We developed an algorithm for this purpose, which is presented within this section. Without loss of generality, we consider the determination of all overlapping subscriptions for a given advertisement in the following descriptions. The calculation of all overlapping

[1]Provided there is no other disjunctive part in the filter expression of the advertisement.

advertisements for a subscription works analogously due to their symmetric definition and internal representation.

In the following subsection (Section 7.2.1), we elaborate on the general overlap calculation approach. Afterwards, we outline the central concept of *disjoint predicates* in Section 7.2.2. This section gradually introduces the determination of disjoint predicates for conjunctive, for disjunctive, and finally for general Boolean advertisements. In Section 7.2.3, we then describe how to calculate the overlapping relationships from the computed disjoint predicates. We conclude in Section 7.2.4 by considering implementation aspects of our approach.

The computation approach we describe in the following subsections is a generic solution to the overlapping task, working for both conjunctive and general Boolean specifications.

7.2.1 General Calculation Approach

Approaches to calculate the overlapping relationships between conjunctive subscriptions and advertisements, for example, sketched in [Müh02], require these specifications to contain at most one predicate per attribute specification. This property of subscriptions and advertisements is then exploited in the computation algorithm: a subscription s overlaps a given advertisement a if, and only if, there exists no non-overlapping, that is, disjoint, predicate in s with respect to a [Müh02]. The conjunctive overlapping calculation algorithm thus counts the number of disjoint predicates per subscription, similar to a conjunctive event filtering approach. We here refer to Section 7.2.2 for details on the notion of disjoint predicates, in particular in conjunction with general Boolean subscriptions and advertisements.

Looking at the semantics of general Boolean subscriptions and advertisements, however, reveals the inapplicability of such a calculation approach: subscriptions and advertisements might contain any number of predicates for a given attribute specification (see Definition 4.3 on page 98 and Definition 7.1 on page 202). Thus the analogous application of the conjunctive calculation approach does not work: Subscriptions and advertisements might overlap even if there exist disjoint predicates between them. That is, the application of conjunctive overlapping calculation approaches in the general Boolean pub-sub model leads to incorrect results.

Evidently, instead of basing the calculation of the overlapping relationships between general Boolean subscriptions and advertisements on conjunctive event filtering approaches, a computation algorithm should be similar to event filtering solutions for general Boolean subscriptions, as proposed in Chapter 4. We take this approach in this chapter and apply a three-step calculation algorithm, firstly determining disjoint predicates, secondly calculating candidate overlapping subscriptions, and thirdly restricting these candidates to overlapping subscriptions. Before proceeding with describing this approach, we introduce the notion of disjoint predicates in combination with general Boolean subscriptions and advertisements.

7.2.2 Disjoint Predicates

We now define our notion of disjoint predicates for a given predicate, that is, a leaf node of an advertisement tree. This definition is extended to general advertisements (and subscriptions) later on.

Definition 7.3 (Disjoint predicates of a leaf node) *One has been given a leaf node n_l of an advertisement tree of advertisement $a_i = (T_i^n, \mathcal{F}_i)$ that contains predicate $p_i = (a_i^n, f_i, op_i)$. A predicate $p_j = (a_j^n, f_j, op_j)$ used in a subscription $s_j = (T_j^n, \mathcal{F}_j)$ is a* disjoint predicate *to n_l, that is, $p_j \in P_{dis}^l(n_l)$, if, and only if:*

1. *predicate p_i refers to the same attribute of the same event type as p_j, that is, $T_i^n = T_j^n$ and $a_i^n = a_j^n$.*
2. *there exists no attribute value a^v that leads to a* true *result if applied to the filter functions f_i and f_j of p_i and p_j, respectively.*

That is, it holds that (a_i^d specifies the domain of attribute a_i^s, see Definition 4.1 on page 96):

$$T_i^n = T_j^n \wedge a_i^n = a_j^n \wedge \nexists a^v \in a_i^d(f_i(a^v, op_i) = f_j(a^v, op_j) = \text{true}).$$

We can compute the set of disjoint predicates for a given predicate based on the applied one-dimensional predicate indexes. This calculation approach is applicable to both directions of computation, subscription-advertisement overlapping and advertisement-subscription overlapping relationships: advertisements need to be indexed in the same way as subscriptions in conjunctive content-based pub-sub systems [Müh02]. In our Boolean pub-sub approach, we adopt this method.

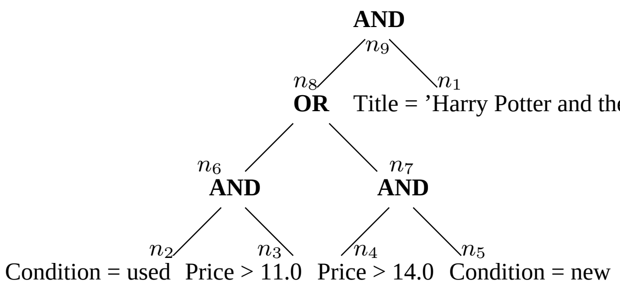


Figure 7.1: Advertisement tree of advertisement a_3 with named inner nodes and leaf nodes.

The calculation of disjoint predicates is based on the filter functions that are used within the predicates of both subscriptions and advertisements. We here refer to our work in [BH06a] for examples on this filter function-based computation approach.

We illustrate our notion of general disjoint predicates in the following example:

Example 7.1 (Disjoint predicates of a leaf node) *For our example advertisement a_3 (see Figure 7.1), the disjoint predicates of its leaf nodes n_1 to n_5 are as follows. To give a simple example, we firstly assume the registration of subscription s_1, that is, $\mathcal{S} = \{s_1\}$:*

$$\begin{aligned}
P^l_{dis}(n_1) &= \varnothing \quad \textit{(phrase "Harry Potter and the Goblet of Fire"} \\
&\qquad \textit{contains phrase "Harry Potter"),} \\
P^l_{dis}(n_2) &= \{p_3\} \ \textit{("used" condition does not fulfill "new" condition),} \\
P^l_{dis}(n_3) &= \varnothing \quad \textit{(prices of more than NZ\$11.00 fulfill prices of less} \\
&\qquad \textit{than NZ\$15.00 and NZ\$12.00),} \\
P^l_{dis}(n_4) &= \{p_5\} \ \textit{(prices of more than NZ\$11.00 do not fulfill a price} \\
&\qquad \textit{less than NZ\$12.00),} \\
P^l_{dis}(n_5) &= \{p_6\} \ \textit{("new" condition does not fulfill "used" condition).}
\end{aligned}$$

Let us now extend this scenario to the registration of several subscriptions, that is, $\mathcal{S} = \{s_1, s_3, s_4\}$[2]. Subscription s_4 has the same specification as subscription s_2 except that predicate p_1^4 is defined as `Title` ∼ "`The Chronicles of Narnia`" (instead of `Title` ∼ "`Harry Potter`"). The disjoint predicates of advertise-

[2]We again use the notation p_j^i to distinguish predicates in the following, stating predicate p_j of subscription s_i.

ment a_3 are as follows:

$$P^l_{dis}(n_1) = \{p^4_1\},\ P^l_{dis}(n_2) = \{p^1_3, p^4_5, p^4_9\},\ P^l_{dis}(n_3) = \{p^4_{11}\},$$
$$P^l_{dis}(n_4) = \{p^1_5, p^4_{11}\},\ P^l_{dis}(n_5) = \{p^1_6, p^4_8, p^4_{12}\}.$$

Based on this notion of disjoint predicates for a given leaf node, we ultimately extend this concept to general Boolean advertisements. For simplicity we initially focus on restricted conjunctive and disjunctive advertisements.

Pure Conjunctive Advertisements

Let us consider a pure conjunctive advertisement a_c with root node n_c. The set of disjoint predicates of n_c, $P^c_{dis}(n_c)$, is the union of the sets of disjoint predicates of each of the child nodes of n_c (i.e., predicates of the tree rooted in n_c).

The reason for this definition is that for all event messages conforming to a conjunctive advertisement a_c, the functions that are given in the predicates of all leaf nodes evaluate to *true*. Hence each predicate that is disjoint to any of these predicates evaluates to *false* on a conforming message. One can thus combine the disjoint predicates for all k children (n_1 to n_k) of n_c (i.e., for all predicates of a_c in this case) to derive these predicates that evaluate to *false*:

$$P^c_{dis}(n_c) \quad = \quad \bigcup_{i=1\ldots k} P^l_{dis}(n_i).$$

Pure Disjunctive Advertisements

Let us now consider a pure disjunctive advertisement a_d with root node n_d. To derive the predicates that are disjoint to n_d, one could build the intersection of all disjoint predicates of the children of n_d. This approach results in a set of those predicates that are disjoint to all predicates of the disjunction. However, predicates that are only disjoint to one or several predicates are neglected in this calculation.

Instead the disjoint predicates of a_d should be expressed by several predicate sets because each disjunctive advertisement a_d contains several descriptions of event messages (involving one predicate each). At least one of these descriptions, that is, predicates, has to evaluate to *true* for each conforming message.

This characteristic of disjunctive advertisements contradicts our previous notion of disjoint predicates as a predicate set. For disjunctions, disjoint predicates should instead be defined as a set of predicate sets. Each of these predicate sets describes one of the options that are expressed by the disjunctive advertisement. That is, for a disjunctive advertisement a_d with root node n_d and k children (n_1 to n_k), it holds that:

$$P^d_{dis}(n_d) \quad = \quad \{P^l_{dis}(n_i) | i = 1 \dots k\}.$$

General Boolean Advertisements

A general Boolean advertisement a might contain both disjunctive and conjunctive operators[3], that is, nodes in the advertisement tree. Thus the disjoint predicates of such an advertisement must also be defined as sets that contain sets of predicates as elements.

The calculation of these disjoint predicates can be based on the operators that are represented by the inner nodes n of advertisement trees. In the following description, we refer to this universal notion of disjoint predicates in combination with any node n of advertisement trees as $P_{dis}(n)$. All elements in set $P_{dis}(n)$ describe one set of disjoint predicates that is induced by n, that is, by that subtree of advertisement a that is rooted in n.

For leaf nodes n_l of general Boolean advertisements a, the calculation algorithm looks up the utilized predicate indexes as introduced in Section 7.2.2. It then embeds the computed disjoint predicates in a set to obtain the refined, universal notion:

$$P_{dis}(n_l) \quad = \quad \{P^l_{dis}(n_l)\}.$$

For a conjunctive node n_c with k children, n_1 to n_k, one unites each set of disjoint predicates of each child of n_c with each disjoint predicate set of all other children:

$$P_{dis}(n_c) \quad = \quad \{\bigcup_{i=1 \dots k} s | s \in P_{dis}(n_i)\}.$$

For a disjunctive node n_d with k children, n_1 to n_k, one unites the computed

[3]As for subscriptions, negations are pushed down in the direction of leaf nodes.

sets of disjoint predicates of all children of n_d:

$$P_{dis}(n_d) \quad = \quad \bigcup_{i=1\ldots k} P_{dis}(n_i).$$

Finally, we define the disjoint predicates of a general Boolean advertisement a as equivalent to the disjoint predicates of its root node n, that is, $P_{dis}(a) = P_{dis}(n)$. Recursively calculating the disjoint predicates of a, ultimately allows for the determination of the overlapping subscriptions for a. We describe this method in the following subsection, following this example:

Example 7.2 (Disjoint predicates of general Boolean advertisements) *Let us again assume the registration of subscription s_1 (see Section 3.3, page 79). The disjoint predicates for the leaf nodes of advertisement a_3 (see Figure 7.1) are straightforwardly derived from our calculations in Example 7.1, page 206:*

$$\begin{aligned}
P_{dis}(n_1) &= \{P^l_{dis}(n_1)\} = \{\varnothing\},\\
P_{dis}(n_2) &= \{P^l_{dis}(n_2)\} = \{\{p_3\}\},\\
P_{dis}(n_3) &= \{P^l_{dis}(n_3)\} = \{\varnothing\},\\
P_{dis}(n_4) &= \{P^l_{dis}(n_4)\} = \{\{p_5\}\},\\
P_{dis}(n_5) &= \{P^l_{dis}(n_5)\} = \{\{p_6\}\}.
\end{aligned}$$

For the conjunctive nodes n_6 and n_7 of a_3, the disjoint predicates are as follows:

$$P_{dis}(n_6) = \{\{p_3\}\}, P_{dis}(n_7) = \{\{p_5, p_6\}\}.$$

And for the disjunctive node n_8, we derive the following disjoint predicates:

$$P_{dis}(n_8) \quad = \quad \{\{p_3\}, \{p_5, p_6\}\}.$$

Finally, for the conjunctive root node n_9, and a_3 itself, the disjoint predicates are the same as for n_8 in this example (because it holds $P_{dis}(n_1) = \{\varnothing\}$):

$$P_{dis}(a_3) \quad = P_{dis}(n_9) = P_{dis}(n_8) = \quad \{\{p_3\}, \{p_5, p_6\}\}.$$

For our extended example scenario with $\mathcal{S} = \{s_1, s_3, s_4\}$, it holds:

$$\begin{aligned}
&P_{dis}(n_1) = \{\{p^4_1\}\},\ P_{dis}(n_2) = \{\{p^1_3, p^4_5, p^4_9\}\},\ P_{dis}(n_3) = \{\{p^4_{11}\}\},\\
&P_{dis}(n_4) = \{\{p^1_5, p^4_{11}\}\},\ P_{dis}(n_5) = \{\{p^1_6, p^4_8, p^4_{12}\}\},
\end{aligned}$$

$$P_{dis}(n_6) = \{\{p_3^1, p_5^4, p_9^4, p_{11}^4\}\},\ P_{dis}(n_7) = \{\{p_5^1, p_6^1, p_8^4, p_{11}^4, p_{12}^4\}\},$$
$$P_{dis}(n_8) = \{\{p_3^1, p_5^4, p_9^4, p_{11}^4\}, \{p_5^1, p_6^1, p_8^4, p_{11}^4, p_{12}^4\}\},$$
$$P_{dis}(a_3) = P_{dis}(n_9) = \{\{p_3^1, p_1^4, p_5^4, p_9^4, p_{11}^4\}, \{p_5^1, p_6^1, p_1^4, p_8^4, p_{11}^4, p_{12}^4\}\}.$$

7.2.3 Overlap Based on Disjoint Predicates

Having introduced the notion of disjoint predicates for any general Boolean advertisement a, we now describe how to determine the overlapping subscriptions based on the calculated disjoint predicates. The approach comprises three steps. The first of these steps, *disjoint predicate matching*, involves the computation of the disjoint predicates $P_{dis}(a)$ of a.

Candidate Overlapping Subscription Matching

The second step, *candidate overlapping subscription matching*, determines a set of candidate subscriptions that potentially overlap the given advertisement a. This set excludes all those subscriptions that definitely do not overlap a. We can base the determination of these candidates on the number of disjoint predicates per subscription s with respect to a, $|P_{dis}(s, a)|$, as described in the following:

For all indexed subscriptions s, the Boolean filtering algorithm determines the minimal number of fulfilled predicates that is required for a fulfilled subscription, $p_{min}(s)$. For candidate overlapping subscription matching, this subscription-specific property is related to the number of disjoint predicates of s with respect to a, $|P_{dis}(s, a)|$, and the overall number of predicates of s, $|P(s)|$. For each candidate subscription s, it has to hold that:

$$|P(s)| \quad \geq \quad |P_{dis}(s, a)| + p_{min}(s). \tag{7.1}$$

This inequality shows that the subscription tree of s, potentially, evaluates to *true* for the given disjoint predicates of a. That is, even if assuming that all disjoint predicates evaluate to *false* (which is the case for an event message conforming to a), subscription s might still match a message that conforms to the given advertisement.

Previously we defined the disjoint predicates of advertisement a, $P_{dis}(a)$, as a set that contains predicate sets: $P_{dis}(a) = \{\{p_i, \ldots, p_j\}, \ldots, \{p_k, \ldots, p_l\}\}$.

Specifically, value $|P_{dis}(s, a)|$ is defined as the maximal cardinality of these sets when only considering predicates p that are included in $s = (T^n, \mathcal{F})$, that is, $p \in \mathcal{P}(\mathcal{F})$.

Final Overlapping Subscription Matching

Having determined a set of candidate overlapping subscriptions as the second step of the algorithm, in the third step— *final overlapping subscription matching*—it remains to identify whether these candidates constitute an overlapping subscription. Similarly to the event filtering approach, the algorithm can obtain this information by evaluating the subscription trees of candidates.

For the evaluation of these trees, the system is not required to evaluate the filter functions of the predicates in leaf nodes. Their assignment is already known from the previous steps: For a disjoint predicate, the respective leaf node is assumed to evaluate to *false*. For all non-disjoint, that is, overlapping, predicates, the algorithm assumes a value of *true*. These assumptions are implied by the facts that (i) disjoint predicates never evaluate to *true* on event messages that conform to advertisement a, and (ii) non-disjoint predicates are potentially fulfilled by messages that conform to a. A candidate subscription constitutes an overlapping subscription if its subscription tree evaluates to *true* for the described assignment of predicates.

Again this general calculation approach has to be customized to our universal definition of disjoint predicates of advertisements a, $P_{dis}(a)$, as being a set of predicate sets. That is, the evaluation of subscription trees needs to be performed for all elements of the disjoint predicate set $P_{dis}(a)$ (being sets of predicates) that distinguish a subscription as a candidate. Hence the subscription tree of subscription s is evaluated for all those elements in $P_{dis}(a)$ that lead to a sum of number of disjoint predicates and minimal number of fulfilled predicates less than or equal to the total number of predicates of s (see Equation 7.1, page 210).

We illustrate these two latter computation steps of the overlapping calculation in the following example:

Example 7.3 (Overlap based on disjoint predicates) *For advertisement a_3 (see Figure 7.1), we determined the set of disjoint predicates as:*

$$P_{dis}(a_3) \quad = \quad \{\{p_3\}, \{p_5, p_6\}\}.$$

For subscription s_1, it holds that:

$$|P(s_1)| = 6,\ p_{min}(s_1) = 4.$$

Thus subscription s_1 is a candidate subscription for both elements in $P_{dis}(a_3)$ because $6 \geq 1{+}4$ (for set $\{p_3\}$) and $6 \geq 2{+}4$ (for set $\{p_5, p_6\}$). Subscription s_1, in fact, is an overlapping subscription because its subscription tree can result in true *if p_3, or p_5 and p_6 evaluate to* false *(it is sufficient if this is the case for either one of these predicate sets).*

For our extended scenario, we determined the set of disjoint predicates as:

$$P_{dis}(a_3) \quad = \quad \{\{p_3^1, p_1^4, p_5^4, p_9^4, p_{11}^4\}, \{p_5^1, p_6^1, p_1^4, p_8^4, p_{11}^4, p_{12}^4\}\}.$$

With respect to s_1, these predicates are the same as the result of our previous calculation; thus subscription s_1 is an overlapping subscription. For the remaining two subscriptions, it holds:

$$|P(s_3)| = 7,\ p_{min}(s_3) = 3,$$
$$|P(s_4)| = 12,\ p_{min}(s_4) = 5.$$

Both s_3 and s_4 are thus candidate subscriptions. It holds $7 \geq 0 + 3$ (none of the sets in $P_{dis}(a_3)$ contains any predicates of s_3), $12 \geq 4 + 5$ (for set $\{p_3^1, p_1^4, p_5^4, p_9^4, p_{11}^4\}$), and $12 \geq 4 + 5$ (for set $\{p_5^1, p_6^1, p_1^4, p_8^4, p_{11}^4, p_{12}^4\}$).

Furthermore, subscription s_3 is an overlapping subscription because its subscription tree can result in true*, for example, if all of its predicates evaluate to* true*. Subscription s_4, however, is no overlapping subscription because its subscription tree results in* false *if either p_1^4, p_5^4, p_9^4, and p_{11}^4 evaluate to* false*, or p_1^4, p_8^4, p_{11}^4, and p_{12}^4 evaluate to* false.

In the following section, we describe the practical implementation of the presented overlapping calculation approach within our prototype BoP.

7.2.4 Implementation of the Calculation Approach

To support the calculation of overlapping relationships for both directions, that is, subscription-advertisement and advertisement-subscription overlap, BoP indexes advertisements in the same way as subscriptions. This approach aligns with current (conjunctive) calculation proposals, using the same index structures for conjunctive advertisements as for conjunctive subscriptions (see [Müh02]). Although in the following descriptions we illustrate the practical realization in BoP from the subscription-advertisement overlapping perspective, BoP supports both directions. In Figure 7.2, we illustrate an outline of the overall algorithm and the involved data structures to allow for a better overview of the approach.

Disjoined Predicates

To designate predicates as disjoint predicates in BoP, we apply the existing implementation of a predicate bit vector, which is already used as fulfilled predicate vector in the filtering process. For all leaf nodes of advertisement trees, BoP directly utilizes the one-dimensional predicate index structures to determine the identifiers of disjoint predicates and stores them in individual *predicate bit vectors* (one vector per predicate, that is, leaf node, of the advertisement).

The disjoint predicate set of the advertisement itself is represented by an array of bit vectors (*disjoint predicate array* of disjoint predicate vectors): for disjunctive nodes of advertisement trees, BoP unites the arrays of their children, for example, the disjoint predicate arrays of two leaf nodes (one entry each) get combined to a disjoint predicate array of size two.

For conjunctive nodes, initially BoP recursively calculates the disjoint predicate array for the first child (intermediate result). Subsequently, it calculates these arrays for the remaining children and, in each case, performs the required union operation with the previously determined intermediate result. This union operation combines each element (i.e., a predicate bit vector) in the array of the intermediate result with each element in the array of the currently processed child. Finally, this process leads to the overall result, that is, the disjoint predicate array, for the conjunctive node. The disjoint predicate matching step is illustrated in the top part of Figure 7.2.

For pure conjunctive advertisements, we integrated an optimization to this approach into BoP that only uses one predicate bit vector for the whole calculation. This optimization thus performs the same operations as specialized conjunctive computation approaches if subscribers and publishers only use restricted conjunctive forms.

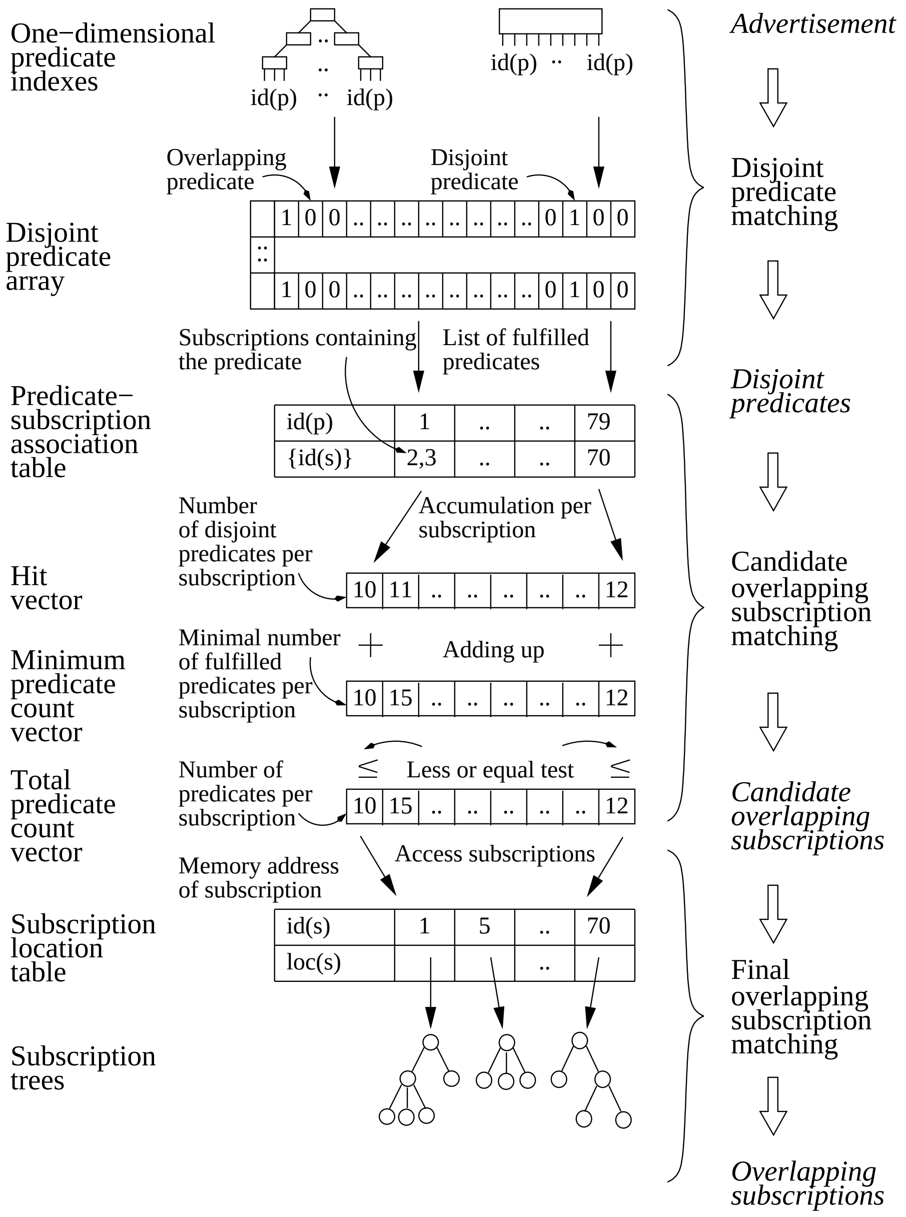

One−dimensional predicate indexes
id(p) ·· id(p)
id(p) ·· id(p)
Overlapping predicate
Disjoint predicate
Disjoint predicate array
1 0 0 .. .. 0 1 0 0
Subscriptions containing the predicate
List of fulfilled predicates
Predicate−subscription association table
id(p) 1 79
{id(s)} 2,3 70
Number of disjoint predicates per subscription
Accumulation per subscription
Hit vector
10 11 12
Minimal number of fulfilled predicates per subscription
Adding up
Minimum predicate count vector
10 15 12
Less or equal test
Number of predicates per subscription
Total predicate count vector
Memory address of subscription
Access subscriptions
Subscription location table
id(s) 1 5 .. 70
loc(s) ..
Subscription trees
Advertisement
Disjoint predicate matching
Disjoint predicates
Candidate overlapping subscription matching
Candidate overlapping subscriptions
Final overlapping subscription matching
Overlapping subscriptions

Figure 7.2: Overview of disjoint predicate matching, candidate overlapping subscription matching, and final overlapping subscription matching in the overlapping calculation algorithm.

Candidate Overlapping Subscriptions

Having calculated the disjoint predicate array, BoP uses the existing hit vector implementation to determine the candidate overlapping subscriptions for an advertisement a: for each bit that is set in one of the disjoint predicate vectors (each vector represents one element in $P_{dis}(a)$), BoP uses the predicate-subscription association table to determine those subscriptions that contain the respective predicate. For each of these subscriptions, the corresponding entry in the hit vector is increased. Having performed this counting for all entries in a disjoint predicate vector, the hit vector states the number of disjoint predicates per subscription for the processed disjoint predicate vector.

If the sum of minimal number of fulfilled predicates for a subscription s, stored in the minimum predicate count vector, and the current entry in the hit vector for s is less than or equal to the total predicate number of s, s is a candidate overlapping (see Equation 7.1). To allow for an efficient determination of candidates, BoP stores the total predicate numbers per subscription in a *total predicate count vector*. We illustrate this candidate overlapping subscription matching process and the involved data structures in the middle part of Figure 7.2.

Overlapping Subscriptions

For the determined candidates, it remains to analyze whether they constitute a real overlap. For this purpose, BoP assigns those values to the leaf nodes of subscription trees that are described in Section 7.2.3: if a bit in the currently processed predicate bit vector is set for the predicate in the leaf node, BoP assigns *false* to this leaf node. For all other predicates, it assigns a value of *true*.

To prevent the multiple evaluation of subscription trees, BoP applies another bit vector, having entries per subscription s, that states whether s was already identified as a real overlap. If this is the case, further evaluations of the subscription tree of s are avoided even if s constitutes a candidate for another disjoint predicate vector in the disjoint predicate array. Additionally, BoP applies a short-circuiting optimization, similar to the one for the general Boolean event filtering algorithm. The lower part of Figure 7.2 illustrates this final overlapping subscription matching process.

Function and Decision Problem

Advertisement-based pub-sub systems can significantly benefit when distinguishing the overlapping calculation as either function or decision problem (see Section 2.1.3, page 20). To solve the function problem, the system needs to determine all overlapping subscriptions for the given advertisement. For the decision problem, however, it is sufficient to determine whether at least one overlapping subscription exists.

BoP includes solutions to both of these problems while the decision problem is solved as a shortcut to the function problem: as soon as an overlapping subscription is determined, BoP discontinues the computation process and returns "yes". It thus avoids the evaluation of various candidate overlapping subscriptions in the final overlapping subscription matching process. Conjunctive algorithms, however, do not benefit to a large extent when solving the decision problem. They still have to process all disjoint predicates in order to find out whether a subscription does not contain any one of them. We refer to Chapter 8 for results of practical experiments showing this behavior.

In practice, content-based pub-sub systems require solutions to both problems under different circumstances. For example, for the decision whether a subscription needs to be forwarded to a particular neighbor, it is sufficient to know whether this neighbor previously forwarded an overlapping advertisement (decision problem). However, if an advertisement is deregistered, the system needs to determine all overlapping subscriptions to decide whether they can be removed from event routing tables (function problem).

Thus it is crucial to solve the decision problem efficiently because it is required when subscriptions are registered. Deregistrations (requiring the function problem to be solved), on the other hand, might be delayed and fully distributed in the overall system once the system load is below a certain threshold. Clients do not recognize these delayed deregistrations because they are merely postponed internally (potentially leading to internal false positives).

7.3 Advertisement-Based Routing Optimization: Advertisement Pruning

Having presented an approach of calculating the overlapping relationships between general Boolean subscriptions and advertisements, we now propose an advertisement-based optimization for pub-sub systems, *advertisement pruning* [BH06c].

The overall idea of this optimization approach is similar to our subscription-based optimization. By applying the optimization, that is, by pruning advertisement trees, the system primarily aims at decreasing the complexity of advertisements (subscription routing entries), with respect to both the space complexity for their storage and the time complexity for their processing in the overlapping calculation. This reduction in memory requirements and increase in efficiency are the target parameters of advertisement pruning.

However, advertisement pruning classifies as interfering optimization because of its alteration of subscription routing entries. Due to the pruning of advertisements, the existing overlapping relationships increase, being the secondary influence of the optimization. Thus advertisement pruning aims at only marginally increasing these overlapping properties.

We formulated the optimization goal of advertisement pruning in Part 2b of our central hypothesis (page 6):

> *Advertisement pruning increases system efficiency and decreases routing table size, while only marginally affecting the existing overlap.*

System efficiency in this hypothesis refers to the efficiency for calculating the overlap, as described before.

We structure the remainder of this section as follows: In Section 7.3.1, we relate the goals of advertisement pruning to subscription pruning and argue why the previously developed ranking measures are inapplicable to the advertisement-based optimization. Then, in Section 7.3.2 we discover what parameters influence the existing overlapping relationships in a pub-sub system. The incorporation of these parameters into a characteristic measure for general Boolean advertisements is then the focus of Section 7.3.3. Finally, Section 7.3.4 develops an applicable ranking measure that is based on the introduced parameters, and Section 7.3.5 elaborates on practical realization issues of advertisement pruning.

7.3.1 Using Subscription Pruning Rankings?

One could expect the ranking measures to select the preferable subscription pruning operation could, at least to a certain extent, be mapped onto the advertisement pruning problem. However, subscription and advertisement pruning, in fact, follow different optimization goals; the previously proposed subscription-based ranking measures are thus not applicable to advertisement pruning:

In practice, the subscription pruning rankings that lead to the best overall optimization are based on the accuracy of subscriptions (see Chapter 8). That is, subscription pruning operations should generalize subscriptions (i.e., event routing entries) as little as possible to only marginally increase the secondary optimization parameter, the network load. The accuracy of subscriptions is modeled by their selectivity, which is estimated on incoming event messages and existing subscription index structures.

When applying advertisement pruning, the pub-sub system aims at increasing the amount of overlap between subscriptions and advertisements as little as possible. This property, at a first glance, is independent of incoming event messages. Nevertheless, the algorithm can partially utilize the existing selectivity information, as shown later on. What is required to effectively prune advertisements is rather a correlation between registered subscriptions and advertisements because advertisement pruning operations, ideally, only slightly alter the existing overlapping relationships, that is, a property between subscriptions and advertisements.

Hence advertisement pruning requires a different ranking approach than the already developed measures (also the memory and the efficiency-based measure do not take advertisements into account).

7.3.2 Influences on Overlap

In this section, we identify the factors that affect the overlapping relationships between subscriptions and advertisements. We then incorporate these factors into an advertisement-specific characterization property in Section 7.3.3 and into a ranking measure that estimates the influences of advertisement pruning in Section 7.3.4.

The proposed algorithm to determine overlapping subscriptions or advertisements utilizes the concept of disjoint predicates for its calculations. The number of disjoint predicates per subscription[4] is then used to compute a set of candidate overlapping subscriptions. Generally the fewer candidates exist,

[4]We again consider the subscription-advertisement overlapping direction for our descriptions, as we did in Section 7.2.

the more efficient the calculation of the overlapping relationships (due to the need to evaluate the subscription trees of these candidates).

Taking this property into account, an advertisement pruning operation should increase the number of disjoint predicates. Such an increase, however, is impossible because each pruning operation removes some predicates of an advertisement and thus the corresponding disjoint predicates. Pruning operations need to aim at removing as few disjoint predicates as possible to, in turn, enlarge the number of candidates as little as possible (Influence 1).

However, if merely considering the number of overlapping candidates, pruning might still strongly increase the existing overlapping relationships. In the worst case, before an advertisement pruning operation, none of the identified candidates constitutes an overlap, but afterwards all candidates do overlap. Therefore, a ranking measure also needs to consider what predicates, and thus what disjoint predicates, are removed due to pruning (Influence 2).

Ideally the removed disjoint predicates do not influence whether a subscription and an advertisement overlap, that is, the remaining disjoint predicates should still disqualify subscriptions and advertisements from overlap. Thus advertisement pruning operations must not remove those predicates from advertisements that lead to disjoint predicates which prevent a non-overlapping subscription from becoming an overlapping one. Making rational assumptions about the usage of predicates, pub-sub systems can partially base pruning decisions on the selectivity of predicates. We elaborate on this proposal in the next subsection and present an advertisement-specific characterization property that, as described later on, helps the system to reach pruning decisions.

7.3.3 Characterizing a Boolean Advertisement

As a preliminary step to developing a ranking measure, we quantify an *overlapping characteristic* for advertisements. Eventually, with the help of this characteristic we can estimate the effects of pruning operations in a ranking measure. This measure allows for the determination of the preferable among all possible advertisement pruning options.

The overlapping characteristic combines the number of disjoint predicates of an advertisement—referred to as *quantitative overlapping characteristic* in the following—with the influence of these predicates on the number of overlapping relationships—the *qualitative overlapping characteristic*. That is, the proposed characteristic incorporates both of the previously identified influences of pruning. The overlapping characteristic for an advertisement a is successively calculated by the system based on the advertisement tree of a.

Leaf Nodes

For leaf nodes n_l of advertisement trees, the quantitative overlapping characteristic includes (i) the number of predicates that are disjoint to the predicate that is stored in n_l, and (ii) the number of subscriptions that specify these disjoint predicates.

To allow for an efficiently computable qualitative overlapping characteristic, let us make the following assumptions about the usage of predicates in subscriptions: generally, predicates of subscriptions have highly dissimilar selectivities. For example, predicates on attribute `Title` or `Author` show a high selectivity value, that is, they are quite restrictive, whereas predicates on attribute `Condition` or a low price have a low selectivity, that is, they are relatively general. In practice, there is the tendency that highly selective predicates strongly determine whether a subscription tree might be fulfilled (with respect to both matching messages and overlapping advertisements), compared to general predicates. Thus the more selective a predicate in subscriptions, the more important its state of fulfillment.

Putting together these observations with the importance of disjoint predicates when evaluating overlapping candidates, pub-sub systems should aim to remove general disjoint predicates rather than highly selective ones. The qualitative overlapping characteristic should thus incorporate the selectivity of disjoint predicates: the higher the selectivity of a disjoint predicate p, the higher the corresponding qualitative overlapping characteristic for p. The obvious reason is that highly selective predicates (in subscriptions) are potentially disjoint to a large number of predicates (in advertisements), whereas general predicates (in subscriptions) are potentially disjoint to a small number of predicates (in advertisements). Thus it is advantageous if pruning operations result in the removal of general disjoint predicates (in subscriptions) because they are likely to only marginally increase the existing overlapping relationships.

We define the overlapping characteristic $ovl(n_l)$ of a leaf node n_l of an advertisement as follows. It always holds $0 \leq ovl(n_l) \leq 1$:

$$ovl(n_l) \;=\; \frac{1}{|predSubAssoc|} \sum_{p_i \in P^l_{dis}(n_l)} \frac{predSubAssoc(p_i)}{\sqrt{|msg_f(p_i)| + 1}}.$$

We also give the pseudo code for an algorithmic realization of this calculation approach in Algorithm 5[5].

Algorithm 5: Overlapping characteristic estimation $ovl^{\approx}(n)$ for leaf nodes
Input: A leaf node n
Output: The estimation $(ovl^{min}(n), ovl^{avg}(n), ovl^{max}(n))$

```
EstimateOverlapping(n)
(1)    arrDisj ← DisjointPredicates(n)
(2)    est ← 0.0
(3)    foreach p in arrDisj
(4)       add ← PredSubAssoc(p)
(5)       add ← add ÷ Sqrt(MatchingMessages(p) + 1)
(6)       est ← est + add
(7)    est ← est ÷ TotalPredSubAssoc()
(8)  return (est, est, est)
```

The expression in the right multiplication factor of $ovl(n_l)$ sums up the overlapping characteristics of all disjoint predicates p of leaf node n_l (determined in Line 1 of the algorithm using the function DisjointPredicates()). The elements in the sum contain a quantitative part $predSubAssoc(p)$ (function PredSubAssoc(p), Line 4), describing the number of predicate-subscription associations of a disjoint predicate p, that is, how many subscriptions contain predicate p.

The qualitative part is given by the denominator, representing the selectivity of p. The number of messages that fulfill the predicate (referred to as $|msg_f(p)|$ in the equation and function MatchingMessages(p) in Line 5 of Algorithm 5) is known from the selectivity estimation for subscriptions. The influence of the qualitative part is lessened by using the square root of the number of matching messages. Taking this approach of weighting quantitative and qualitative overlapping characteristic has led to good results in empirical studies.

The left coefficient of $ovl(n_l)$ ensures that the overlapping characteristic is always between 0 and 1. Value $|predSubAssoc|$ describes the total number of predicate-subscription associations (function TotalPredSubAssoc(), Line 7). The case $ovl(n_l) = 1$ occurs if a leaf node n_l has all registered predicates (from subscriptions) as disjoint predicates with a selectivity of zero each.

[5]We provide this pseudo code for reasons of completeness. We also give the pseudo code for other, non-leaf nodes of advertisement trees.

This definition of an overlapping characteristic assigns high values to leaf nodes (including predicates) that show a large number of disjoint predicates. These disjoint predicates are weighted according to their selectivity, that is, their importance in disqualifying a candidate subscription from being an overlapping subscription.

For the Boolean operators in advertisement trees, the algorithm estimates the overlapping characteristic. This estimation $ovl^{\approx}(n)$ for nodes n contains three values, the *minimal possible overlapping characteristic*, the *average overlapping characteristic*, and the *maximal possible overlapping characteristic*:

$$ovl^{\approx}(n) \quad = \quad (ovl^{min}(n), ovl^{avg}(n), ovl^{max}(n)).$$

For a leaf node n_l, these three estimations have the same value, the overlapping characteristic $ovl(n_l)$ as defined previously:

$$ovl^{min}(n_l) = ovl^{avg}(n_l) = ovl^{max}(n_l) = ovl(n_l).$$

Before considering nodes of advertisements other than leaves, we describe the calculation of the overlapping characteristic for these leaves in the following:

Example 7.4 (Overlapping characteristic for leaf nodes) *Let us consider the only registration of example advertisement a_3 (see Figure 7.1) and example subscription s_1 (see Section 3.3). It holds $|predSubAssoc| = 6$, as well as $predSubAssoc(p_i) = 1$ for $i = 1 \ldots 6$. Additionally, we assume the following values for the number of matchings for some predicates of s_1:*

$$|msg_f(p_3)| = 500, \ |msg_f(p_5)| = 1,500, \ |msg_f(p_6)| = 2,000.$$

For the five leaf nodes n_1 to n_5 of a_3 then:

$$\begin{aligned}
ovl(n_1) &= \frac{1}{6} \times 0 = 0, \\
ovl(n_2) &= \frac{1}{6} \times (\frac{1}{\sqrt{500+1}}) \approx 0.00745, \\
ovl(n_3) &= \frac{1}{6} \times 0 = 0, \\
ovl(n_4) &= \frac{1}{6} \times (\frac{1}{\sqrt{1,500+1}}) \approx 0.00430, \\
ovl(n_5) &= \frac{1}{6} \times (\frac{1}{\sqrt{2,000+1}}) \approx 0.00373.
\end{aligned}$$

These results show that nodes n_1 and n_3 have the least significance in deter-

mining candidate subscriptions and their state of overlap, that is, n_1 and n_3 do not have any disjoint predicates in this example. This is followed by node n_5 and n_4. The most significant indicator for candidate subscriptions and their overlapping properties is node n_2. These results align with our assumptions about predicates: The higher the selectivities of disjoint predicates, the more important they are for restricting the overlapping relationships. In this case, the disjoint predicate of node n_2 (predicate p_3) is the most selective one.

Conjunctive Nodes

For conjunctive nodes n_c, the algorithm takes an estimation approach for the overlapping characteristic calculation. We decided to proceed in this way due to the lack of efficiently-computable information about the relationships among disjoint predicates. The estimation approach, more importantly, allows for a time- and space-efficient calculation of the required overlapping characteristic. For conjunctive nodes, both concepts of the overlapping characteristic, the qualitative and the quantitative part, are included in the computation process.

We give the pseudo code for the computation in Algorithm 6. The algorithm walks through all children of a conjunctive input node (Line 4 in Algorithm 6), recursively estimates the overlapping characteristics for these children (Line 5), and finally combines these results with the previously known, intermediate estimation (Lines 6 to 8).

The minimal possible overlapping characteristic $ovl^{min}(n_c)$ occurs if all disjoint predicates are shared among the children of the conjunctive node n_c. It is thus the maximal value of the overlapping characteristic of all children (Line 6).

The average overlapping characteristic $ovl^{avg}(n_c)$ approximates a mean value for the characteristic estimations of the children. It describes the expected mean if assuming independent child nodes and an equiprobable distribution of disjoint predicates among these children (Line 7).

Finally, the maximal possible overlapping characteristic $ovl^{max}(n_c)$ occurs if the disjoint predicates of child nodes exclude each other. It is thus, at most, the sum of the characteristic estimations of all children (Line 8) but further restricted to not increase over 1 (Line 9).

We illustrate the calculation of the overlapping rank for conjunctive nodes in the following example:

Algorithm 6: Overlapping characteristic estimation $ovl^{\approx}(n)$ for conjunctive nodes
Input: A conjunctive node n
Output: The estimation $(ovl^{min}(n), ovl^{avg}(n), ovl^{max}(n))$

```
EstimateOverlapping(n)
(1)   min ← 0.0
(2)   avg ← 0.0
(3)   max ← 0.0
(4)   foreach c in n.children
(5)       e ← EstimateOverlapping(c)
(6)       min ← Max(min, e.min)
(7)       avg ← avg + e.avg − (avg × e.avg)
(8)       max ← max + e.max
(9)   if max > 1.0
(10)      max ← 1.0
(11)  return (min, avg, max)
```

Example 7.5 (Overlapping characteristic for conjunctive nodes) *Let us assume the setting that is given in Example 7.4 (page 222). The calculation of the overlapping characteristic for the two conjunctive nodes n_6 and n_7 of advertisement a_3 is as follows:*

$$\begin{aligned}
ovl^{min}(n_6) &= \max(0.00745, 0) = 0.00745,\\
ovl^{avg}(n_6) &= 0.00745 + 0 - (0.00745 \times 0) = 0.00745,\\
ovl^{max}(n_6) &= \min(1.0, 0.00745 + 0) = 0.00745,\\
ovl^{min}(n_7) &= \max(0.00430, 0.00373) = 0.00430,\\
ovl^{avg}(n_7) &= 0.00430 + 0.00373 - (0.00430 \times 0.00373) \approx 0.00801,\\
ovl^{max}(n_7) &= \min(1.0, 0.00430 + 0.00373) = 0.00803.
\end{aligned}$$

Disjunctive Nodes

For the second kind of inner node of advertisement trees, disjunctions n_d, our algorithm also applies an estimation approach to determine the overlapping characteristic. This characteristic again contains a qualitative and a quantitative part, and is based on a combination of the overlapping characteristics of the children of n_d. The pseudo code of the algorithm to derive the overlapping characteristic $ovl^{\approx}(n_d)$ is given in Algorithm 7.

The minimal possible characteristic $ovl^{min}(n_d)$ is described by the characteristic estimation of that child node of disjunction n_d that has the smallest overlapping characteristic (Line 6 in Algorithm 7). This definition is grounded in the fact that, under all circumstances, this value of the overlapping characteristic does hold, independently of what part (that is, child node) of the disjunctive node leads to the overlap.

Similar to conjunctive nodes, the average overlapping characteristic of disjunction n_d, $ovl^{avg}(n_d)$, considers child nodes independently of each other and assumes an equiprobable distribution of disjoint predicates among its children (Line 7).

The maximal overlapping characteristic $ovl^{max}(n_d)$ describes the situation that all child nodes of the disjunction are fulfilled for conforming messages and that their disjoint predicates exclude each other (Line 8).

Algorithm 7: Overlapping characteristic estimation $ovl^{\approx}(n)$ for disjunctive nodes
Input: A disjunctive node n
Output: The estimation $(ovl^{min}(n), ovl^{avg}(n), ovl^{max}(n))$

```
EstimateOverlapping(n)
(1)   min ← 1.0
(2)   avg ← 0.0
(3)   max ← 0.0
(4)   foreach c in n.children
(5)      e ← EstimateOverlapping(c)
(6)      min ← Min(min, e.min)
(7)      avg ← avg + e.avg − (avg × e.avg)
(8)      max ← max + e.max
(9)   if max > 1.0
(10)     max ← 1.0
(11)  return (min, avg, max)
```

We show the calculation of the overlapping characteristic for disjunctive nodes in the following:

Example 7.6 (Overlapping characteristic for disjunctive nodes) *Let us again assume the setting that is given in Example 7.4 and 7.5. The calculation of the overlapping characteristic for the disjunctive node n_8 of advertisement a_3 is as follows:*

$$\begin{aligned} ovl^{min}(n_8) &= \min(0.00745, 0.00430) = 0.00430, \\ ovl^{avg}(n_8) &= 0.00745 + 0.00801 - (0.00745 \times 0.00801) \approx 0.0154, \\ ovl^{max}(n_8) &= \min(1.0, 0.00745 + 0.00803) = 0.01548. \end{aligned}$$

This result finally leads to the overlapping characteristic for the root node n_9 of a_3:

$$\begin{aligned} ovl^{min}(n_9) &= \max(0.00430, 0) = 0.00430, \\ ovl^{avg}(n_9) &= 0.0154 + 0 - (0.0154 \times 0) = 0.0154, \\ ovl^{max}(n_9) &= \min(1.0, 0.01548 + 0) = 0.01548. \end{aligned}$$

Using the previous definitions, we finally define the overlapping characteristic of an advertisement a based on the root node n of its advertisement tree, that is, $ovl^{\approx}(a) = ovl^{\approx}(n)$.

Having this means to quantify a characteristic measure for advertisements, which represents their state of overlap, a pub-sub system can determine the effect of advertisement pruning operations, as shown in the following subsection.

7.3.4 Estimating the Influences of Pruning Operations

As with subscription pruning, an important question is: Given a set of registered advertisements, what is the order of advertisement pruning operations to perform? That is, the system firstly needs to determine the preferred pruning operation for each advertisement. Secondly, it needs a means to compare pruning operations of different advertisements to each other.

As identified in Section 7.3.1, advertisement pruning should minimally affect the number of overlapping subscriptions for the pruned advertisement. Because the presented overlapping characteristic estimates a measure for this relationship, a pruning operation should minimally change, that is, decrease, the value of the overlapping characteristic for the respective advertisement.

To describe the influence of pruning, we should apply a proportional measure. This approach helps to weight an absolute change in the overlapping characteristic higher for a small existing characteristic value than for a large one. That is, if there is only a small number of disjoint predicates for a given advertisement, the influence of removing some of them on the existing overlapping relationships is higher than for removing the same number of predicates from an overall large number of disjoint predicates[6].

[6]For simplicity, we ignored the qualitative part of the characteristic in this statement.

We refer to our measure of the influence of a pruning of an advertisement a_i to a_j (based on the overlapping characteristic) as *overlapping characteristic degradation*, $\Delta^{\approx}_{ovl}(a_i, a_j)$. This degradation is defined as follows:

$$\Delta^{\approx}_{ovl}(a_i, a_j) = \max\Big(\frac{ovl^{min}(a_i) - ovl^{min}(a_j)}{ovl^{min}(a_i)}, \frac{ovl^{avg}(a_i) - ovl^{avg}(a_j)}{ovl^{avg}(a_i)}, \frac{ovl^{max}(a_i) - ovl^{max}(a_j)}{ovl^{max}(a_i)}\Big).$$

This definition weights the change in the overlapping characteristic proportionally to the existing characteristic value before performing any pruning, that is, advertisement a_i refers to the originally registered advertisement. Proceeding in this way allows for the incorporation of the overall effects of advertisement pruning even if several pruning operations on the same advertisement are performed in a row.

The final step for the algorithm is to relate all possible pruning operations to each other in order to determine an order of pruning. We elaborate on this issue as well as further implementation-related pruning questions in the following subsection. Firstly, however, we give an example of calculating the overlapping characteristic degradation:

Example 7.7 (Calculation of overlapping characteristic degradation) *We again assume the setting that is given in Examples 7.4 to 7.6. In the following descriptions, we use two indices for the nodes n_i^j of advertisement trees, describing node n_i of advertisement a_j.*

The original advertisement a_3 with root node n_9^3 leads to the following estimated overlapping characteristic (see Example 7.6):

$$ovl^{\approx}(a_3) = (0.0043, 0.0154, 0.01548).$$

Let us assume the removal of node n_1^3 of a_3, leading to a_{3a}. The root node of a_{3a} is n_8^{3a}, describing the same subtree as in a_3. It thus holds $ovl^{\approx}(a_{3a}) = ovl^{\approx}(n_8^{3a})$ but also $ovl^{\approx}(n_8^{3a}) = ovl^{\approx}(n_9^3)$ (see Example 7.6). This leads to:

$$\Delta^{\approx}_{ovl}(a_3, a_{3a}) = \max(0, 0, 0) = 0.$$

Removing node n_4^3, which results in a_{3b}, leads to the following:

$$ovl^{\approx}(a_{3b}) \quad = \quad ovl^{\approx}(n_8^{3b}) = ovl^{\approx}(n_9^{3b}) = (0.00373, 0.0112, 0.0112).$$

This results in an overlapping characteristic degradation $\Delta_{ovl}^{\approx}(a_3, a_{3b})$ of:

$$\Delta_{ovl}^{\approx}(a_3, a_{3b}) = \max(0.133, 0.273, 0.276) = 0.276.$$

Another pruning option, the removal of n_2^3 that is resulting in a_{3c}, leads to this estimation:

$$ovl^{\approx}(a_{3c}) \quad = \quad ovl^{\approx}(n_8^{3c}) = ovl^{\approx}(n_9^{3c}) = (0.0043, 0.00801, 0.00803).$$

The overlapping characteristic degradation $\Delta_{ovl}^{\approx}(a_3, a_{3c})$ is then:

$$\Delta_{ovl}^{\approx}(a_3, a_{3c}) = \max(0, 0.48, 0.481) = 0.481.$$

Hence, if only assuming these three pruning options, the pruning of n_1^3 does not have any influence on the overlapping relationships. This is followed by the pruning of n_4^3 and n_2^3.

7.3.5 Practical Advertisement Pruning

Due to the connection between subscription and advertisement pruning, most implementation considerations of the subscription-based optimization can be simultaneously applied to the advertisement-based option. However, the application of advertisement pruning might lead to different consequences than subscription pruning.

Advertisement Pruning Variants

In Chapter 6 we identified three realization variants of pruning optimizations: post-pruning, pre-pruning, and combined pruning (see Section 6.5, page 188). All of these options are applicable to advertisement pruning as well, leading to similar practical implications. These consequences are amended by the purpose of advertisements, acting as subscription routing entries. However, the general idea of the three realization variants remains: for post-pruning, brokers individually optimize advertisements whenever a neighbor broker forwards an advertisement. Pre-pruning, on the other hand, prunes advertisements before forwarding them to neighbor brokers. Finally, the hybrid, combined pruning, potentially prunes advertisements before and after forwarding them.

An effect of advertisement pruning is to alter, that is, to increase, the existing overlapping relationships between subscriptions and advertisements. Because advertisements determine the forwarding of subscriptions, this alteration, potentially, increases the number of forwarded subscriptions, and thus the number of event routing tables entries and the event routing table size. These additionally forwarded subscriptions are thus false positives with respect to subscription routing.

When using pre-pruning, advertisements are always forwarded in a pruned way. Hence all subscription routing decisions of non-local brokers are based on broadened advertisements, potentially creating false positives. For post-pruning, however, only those subscription routing decisions that are based on pruned advertisements show this property. Subscriptions that are routed based on unpruned advertisements, on the other hand, do not create false positives. Combined pruning, again, unites the implications of both variants.

Advertisement Pruning Structures

Advertisement pruning uses the selectivity information about predicates in its overlapping characteristic measure. This information can be found in the selectivity table, required for subscription pruning. When applying bulk pruning to advertisements, our prototype BoP applies a degradation queue to allow for the efficient determination of the preferred order among possible pruning operations. Changes in the overlapping characteristic measure can be resolved by using the three strategies that were presented in Section 6.6.2.

Deregistrations

Deregistrations of advertisements when applying pruning are supported in the same way as in the un-optimized setting. Due to the reduction of the complexity of advertisements when pruning and the resulting release of index structures (see Section 6.6.3), the removal process from the applied advertisement and predicate indexes works even more efficiently compared to un-optimized routing.

The deregistration of an advertisement, potentially, results in the removal of non-local subscriptions. The application of advertisement pruning does not influence this process. If pruning led to the introduction of false positives, these false positives are removed if the pruned advertisement is deregistered (provided the respective subscription does not overlap another advertisement from the same neighbor as well). False negatives, on the other hand, are never introduced when pruning advertisements.

7.4 Related Work

After having presented how we support general Boolean advertisements in BoP, we now relate our algorithms to existing work. In Section 7.4.1, we investigate general advertisement-based approaches of other pub-sub systems. Section 7.4.2 then specifically covers related advertisement-based optimizations.

7.4.1 Advertisement-based Approaches

The application of advertisements is proposed in conjunction with some content-based pub-sub systems. All of these systems only support conjunctive subscriptions. Advertisements are also defined as conjunctions, or they only specify the message type that is sent by the publisher later on. An example of this type-based approach is Hermes [Pie04]. Content-based pub-sub systems supporting conjunctive advertisements include A-mediAS [Hin03], Padres [LHJ05], Rebeca [Müh02], Siena [CRW99, CRW01], and the proposal in [Hei05].

The algorithms to compute the overlapping relationship, if given at all, are dedicated to the restricted conjunctive forms of advertisements and subscriptions, for example, as described in [Müh02]. These given algorithms cannot be applied to more general subscriptions and advertisements than conjunctive ones, as we argued in detail in Section 7.2.1.

To only base advertisements upon the published event type is clearly less expressive than allowing publishers to further restrict their potentially sent messages by either general Boolean or conjunctive combinations of predicates. Therefore, the mechanisms offered by Hermes [Pie04] do not minimize the number of forwarded subscriptions (and thus the load in brokers) to the same extent as more expressive types of advertisements. However, the overlapping relationship is more efficiently calculated in this case.

The differences between supporting conjunctive and general Boolean expressions, that is, the influences of the required canonical conversion, were outlined in detail in Section 2.6 (page 58). With respect to memory requirements, we can directly apply our characterization framework from Chapter 5 because advertisements need to be indexed in the same way as subscriptions do [Müh02]. With respect to the time efficiency properties of the calculation of overlapping relationships, we present results of an empirical evaluation in Chapter 8.

7.4.2 Advertisement-based Optimizations

We are not aware of any advertisement-tailored optimizations in the existing literature on content-based pub-sub systems. Instead, subscription-based optimizations are applied to advertisements as well. We here refer to Section 2.5 (page 47) for details about these approaches. These solutions share the same drawbacks and have the same assumptions as their subscription-based originals.

Imperfect merging does not make such strong assumptions (see our description in Section 2.5.3) and may have a higher optimization potential than perfect merging [WQV$^+$04]. The work in [LHJ05] presents an approach to improve imperfect merging for subscriptions by incorporating knowledge from advertisements. However, there are no existing approaches that are tailored to optimize advertisements.

As a continuation of our discussion in Section 2.5, these facts describe the general problem of existing optimizations for advertisements: they are either employed independently of their area of use, that is, optimizations do not exploit whether they are applied to subscriptions or advertisements; or the optimizations were specifically developed for subscriptions and cannot be successfully applied to advertisements.

As a result, meaningful evaluations of advertisement optimizations can hardly be found in the existing literature. Siena [CRW01] supports subscription and advertisement covering in its routing protocols. However, this work does not answer the question of the influence of advertisement covering on any system parameter. The same holds for Hermes [Pie04] where the approach only supports little expressive type-based advertisements.

Some other analyses of pub-sub systems consider the existence of advertisements and evaluate the influence of optimizations based on subscriptions on the routing load: Rebeca [Müh02] only analyzes the application of subscription covering and subscription merging in combination with advertisement forwarding. The Padres project [LHJ05], presenting a novel computation approach for covering and merging, also does not consider the optimization of advertisements in its evaluation. We close this gap within this dissertation with an evaluation in the following chapter that directly investigates the influence of advertisement pruning.

7.5 Summary

This chapter provided the final milestone to fully support the general Boolean pub-sub model. We presented how to integrate general Boolean advertisements into BoP and how to solve the problems that arise.

We firstly described our approach to calculate the overlapping relationships between general Boolean subscriptions and advertisements. Similarly to the filtering algorithm, this overlapping calculation takes a three-step computation approach. The proposed algorithm is also applicable to pure conjunctive systems and solves the overlapping task in this case without any overhead compared to specialized conjunctive solutions.

The second part of this chapter focused on advertisement-based routing optimizations. Our proposal, advertisement pruning, is the first advertisement-based optimization that is specifically tailored to optimize these specifications of publishers. It extends our subscription-based optimization and directly aims at maintaining the existing overlapping relationships in the system. Taking this approach keeps the accuracy of subscription routing tables, if this is possible.

Having presented solutions to various tasks in content-based pub-sub systems, we present the results of an extensive empirical evaluation of the implementation of our algorithms within BoP in the following chapter.

Chapter 8

Experimental Evaluation

After having introduced several algorithms to support the general Boolean pub-sub model, we present the results of an empirical evaluation of BoP in this chapter. The overall goal of this analysis is to show the usefulness and applicability of our proposals for scenarios that involve general Boolean subscriptions or advertisements.

In this chapter we focus on the evaluation of BoP as a distributed system; an analysis of filtering in its central broker components was already undertaken in Chapter 5. We briefly summarize the results as follows: our Boolean filtering approach (Chapter 4) allows for a more efficient filtering of general Boolean subscriptions than a general-purpose conjunctive solution in combination with the required canonical conversion. Additionally, our filtering solution is more space-efficient than the conjunctive algorithm, even if subscriptions contain only one disjunction.

Relating these results to our central hypothesis (Section 1.3, page 6), in Chapter 5 we proved Part 1 of this hypothesis. We now take the remaining steps to validate the second part of our hypothesis by successively evaluating our solutions for distributed pub-sub systems.

The structure of this chapter is as follows: In Section 8.1, we specify the general testbed for the subsequently presented analyses and describe the measured parameters. Section 8.2 starts the evaluation of our experimental study by comparing an un-optimized Boolean version of BoP to a conjunctive approach. Our subscription-based optimization, subscription pruning, is initially analyzed in Section 8.3. In Section 8.4 we then investigate the behavior of subscription pruning under varying degrees of cover, and correlate pruning to the covering optimization. Section 8.5 shifts the viewpoint to advertisements, and analyzes and compares our approach to calculate the overlapping relationships between subscriptions and advertisements. Finally, in Section 8.6, we investigate our advertisement-based optimization, advertisement pruning.

8.1 General Experimental Setup

We again use the online auction scenario, as introduced in Chapter 3, within the experimental analysis in this chapter. The setup for this scenario extends the setup for our evaluation of central broker components in Section 5.7.1 (page 142). The results are obtained by using the following parameters: $A_{prop} = 5$, $p^{A}_{mult} = 0.1$, and $p^{T}_{mult} = 0.01$ (see Section 3.2.2, page 76). We variably assign parameter B_{prop} in the following experiments within a range from $B_{prop} = 0.01$ to $B_{prop} = 100$; the default assignment is $B_{prop} = 100$ if not stated otherwise. These different parameter settings result in varying degrees of cover among subscriptions.

Detailed specifications of the realization of attribute domains in our experimental prototype BoP are given in Appendix B.1 (page 321). The exact ranges of operands in subscriptions and advertisements are given in Appendix B.2 (page 322) and Appendix B.3 (page 323), respectively.

For all experiments in this chapter, we use a combined test setting that involves subscriptions of all three subscription classes (see Section 3.3, page 79) and advertisements of all eight advertisement classes (see Section 3.4, page 84). This scenario represents an average workload that combines the characteristics of our example subscription and advertisement classes.

8.1.1 System and Network

Within most of our experiments we analyze a true system setting instead of a simulated one. This limits the scale of the network we can evaluate but, more importantly, has various advantages (see below). In most of the experiments we restrict the network size to five broker components, using the machine configuration described in Section 5.7.1 (see page 142, e.g., 512 MB of RAM and 2 GHz processor). In the experiments for this network scale, we run exactly one broker on each of the physical test machines, connected by a 10 Mbps network. We then undertake selected experiments where we scale the size of the broker network. For this case, we host up to 100 brokers on one physical machine.

The main advantage of analyzing a true system setting is that we can evaluate the real costs of the pub-sub system, for example, its overall efficiency properties, while taking into account various real-world influences, for example, the actual memory requirements of algorithms and their cache behavior. A simulated setting, on the other hand, does not allow for the measurement of these real costs; we thus do not consider such costs in our simulation. In the simulation, we evaluate general parameters, for example, the sizes of routing tables and the created network load.

In a true system analysis there exist restrictions on the size of the evaluated distributed system. To analyze the influence of the network scale on our algorithms, we run experiments on larger networks in Section 8.3. Furthermore, to validate the generality of the results, we show the behavior of our algorithms for the two most extreme network topologies (line topology and star topology) with a growing network scale.

According to the identified quality measures (see Section 2.2, page 23), the event delivery task is not included into the efficiency analysis. Thus the workload in our experiments is independent from the actual subscribers, but rather dependent on the number and characteristics of registered subscriptions. The local brokers of registered subscriptions are uniformly distributed among the overall pub-sub system within our experiments.

8.1.2 Measures and Characteristic Parameters

We analyze the parameters network load, efficiency, and memory requirements using different measures. In this section, we define the applied measures and describe our methodology to derive them. Table 8.1 contains a compact summary of all of these measures.

As well, we introduce characteristic parameters that are used to quantify important attributes of evaluated scenarios in this section. Table 8.2 presents them in a compact form.

Network Load Measure

The *proportional network load*, N, is given by the number of non-local broker components that filter one event message on average, proportional to the overall number of non-local brokers in the network[1]. We explicitly only consider

[1]We only use acyclic topologies, see Section 2.1.2 (page 17).

Table 8.1: Overview of our measures for network load, efficiency, and memory requirements in experiments.

Symbol	Measure name	Range
N	Proportional network load	$0 \leq N \leq 1$
E	Absolute Filter efficiency (in milliseconds)	$E > 0$
E^{prop}	Proportional filter efficiency	$E^{prop} > 0$
E^{fct}	Efficiency of solving the overlapping function problem (in milliseconds)	$E^{fct} > 0$
E^{dec}	Efficiency of solving the overlapping decision problem (in milliseconds)	$E^{dec} > 0$
M	Proportional reduction in memory requirements (subscriptions)	$0 \leq M < 1$
M^{abs}	Absolute memory requirements per non-local routing entry (in byte)	$M^{abs} > 0$
M^{adv}	Proportional reduction in memory requirements (advertisements)	$0 \leq M^{adv} < 1$

non-local brokers in this measure, because event messages are filtered by their local broker under all circumstances.

Measure N always lies between 0 and 1. The value $N = 0$ means that event messages are not forwarded by their local brokers, that is, local brokers filter out all messages. $N = 1$ means that each message is filtered by all brokers, that is, event messages are flooded within the whole system.

Using the number of non-local brokers as a basis for the network load measure directly describes the created network load and its potential increase due to pruning: in the analyzed acyclic networks, every forwarding of message e delivers e to a new broker. For example, if N increases from $N = 0.1$ to $N = 0.2$, that is, a factor of two, the number of non-local brokers that filter a message has doubled. Therefore, the internal network load has doubled.

Ideally the network load N does not increase in the course of pruning. The value of N in an un-optimized system scenario depends on the setting, including network topology and size, and registered subscriptions. Our measure N does not consider the distribution of the actual network load. This distribution depends on the patterns of subscribers and publishers, and the utilized network topology. In our evaluation we assume a uniform distribution of pub-

Table 8.2: Overview of the characteristic parameters in experiments.

Symbol	Characteristic parameter	Range
pru^{prop}	Proportional pruning	$0 \leq M < 1$
cov^{prop}	Covering proportion	$0 \leq cov^{prop} < 1$
ovl^{prop}	Overlapping proportion	$0 \leq ovl^{prop} \leq 1$
$cand^{prop}$	Candidate proportion	$cand^{prop} \geq 0$
$\|B\|$	Number of brokers	$5 \leq \|B\| \leq 100$
$\|s\|$	Registered subscriptions	$100,000 \leq \|s\| \leq 500,000$

lishers and subscribers (see Section 8.1.1), and analyze the overall network load N for two extreme topologies.

Efficiency Measures

For filter efficiency in the distributed system[2] we define two measures, an absolute measure and a proportional measure. For overlapping efficiency we define two absolute measures, one for the overlapping function problem and one for the overlapping efficiency problem.

Absolute Filter Efficiency. We give the *absolute filter efficiency* of the distributed system, E, as the average time that is required to process one event message. This processing includes event filtering and event routing. Small values of E are preferable, that is, the smaller E the more efficient the system. E is derived by publishing large numbers of event messages (always more than 100,000 messages), leading to stable averages; the local brokers of these messages are uniformly distributed over the pub-sub network.

Our experimental methodology is as follows: at the starting point of the measurements, event messages are published (one after the other) at their respective local brokers. The exact starting point is when the first message is filtered by a broker. As argued before, we do not model publishers in our experiments. Messages are put into an incoming message queue by a separate process on each machine that hosts a broker (one could consider this process as

[2]Note that we consider the overall distributed system now and not only its central broker components, as we did in Chapter 5.

the publisher)[3]. Brokers extract messages from this queue, filter them, possibly forward them, and execute notifications. These notifications do not notify subscribers. The event delivery task is explicitly excluded from the analysis (see Section 8.1.1); each notification merely updates statistical information on the broker side.

The end point of a measurement is reached if all brokers have processed all messages in their incoming message queues (including the messages forwarded by neighbor brokers). Dividing the difference in system time between this end point and the start point by the overall number of processed messages (i.e., the number of messages originally put into the queues) leads to E, describing the average amount of time units per message.

Proportional Filter Efficiency. When scaling the network and hosting several brokers on a physical machine, we apply the *proportional filter efficiency* measure, E^{prop}. This measure gives an approximation of the filter efficiency for larger networks. We use E^{prop} when analyzing event routing optimizations; E^{prop} is given proportional to the un-optimized filter efficiency.

For example, $E^{prop} = 0.5$ describes that messages are filtered in half of the original time (advantageous system behavior). A value of $E^{prop} = 2.0$ describes that the filtering process takes double the original time (disadvantageous system behavior). Our methodology to derive the actual filtering times is the same as we described for the absolute filter efficiency measure E.

Using a proportional measure allows us to abstract from the fact that several brokers are hosted on one machine and the implied degrading effect on total filtering time. However, one should keep in mind that individual physical brokers might lead to different results than the sharing of physical machines by several brokers (highly loaded brokers might get scheduled more processing time). Measure E^{prop}, however, gives an indication of the expected filter efficiency.

Overlapping Efficiency Measures. Our measures for the efficiency of an overlapping calculation algorithm are given by the average time to solve the overlapping function problem, the *overlapping function problem efficiency E^{fct}*, and the average time to solve the overlapping decision problem, the *overlapping decision problem efficiency E^{dec}*. These averages are determined by consider-

[3]As long as this process is active, both this process and the broker process share a physical machine.

ing large numbers of registered subscriptions (always more than 100,000) and advertisements (always more than 15,000), leading to stable results.

Our methodology to determine E^{fct} and E^{dec} is to register all subscriptions with the system, measure the system time, solve the overlapping function or decision problem for all given advertisements by applying the respective algorithm, and measure the system time again. The required time span is divided by the total number of advertisements to derive the average value.

Memory Requirements Measures

In the distributed system setting we are interested in deriving statements about the sizes of routing tables. To quantify these sizes for event routing tables, we apply a proportional and an absolute measure. For subscription routing tables, we only require a proportional measure.

Reduction in Event Routing Table Size. When evaluating subscription pruning, the memory requirements measure, M (*proportional reduction in event routing table size*), describes the reduction in event routing table size[4].

The main influence on the event routing table size when pruning is the number of predicates[5] that are indexed by the system, that is, the number of predicate-subscription associations. They determine the size of both predicate-subscription association table and subscription trees. Both of these structures are altered when pruning subscriptions. The sizes of the applied one-dimensional predicate indexes also depend on the predicate numbers but are additionally determined by predicate commonality.

Our measure M expresses the change in these predicate-subscription associations. Subscription pruning merely alters non-local routing table entries; M only includes predicates from non-local routing entries, that is, M describes the proportion of non-local predicate-subscription associations that is removed from the system. $M = 0$ holds in an un-optimized system. Generally, the higher M, the more memory is freed. For example, a value of $M = 0.4$ means that 40 percent of the original non-local predicate-subscription associations are removed (due to pruning). A value of $M = 1$ can never be reached in

[4]We specifically decided to apply this high-level measure in our experiments: the analysis of central brokers components in Section 5.5 (page 132) already investigated the real memory requirements for indexing subscriptions in a Boolean (and a conjunctive) approach.

[5]We do not refer to unique predicates here.

practice because each subscription has to retain at least one of its predicates (i.e., $M < 1$).

Size of One Event Routing Table Entry. Measure M does not lead to comparable statements about the event routing table sizes when comparing Boolean and conjunctive systems. The reason for this mismatch is that both system approaches show different initial routing table sizes, that is, the basis of measure M is different.

To derive comparable results, we can easily apply our characterization framework from Chapter 5 to identify the real routing table size for both Boolean and conjunctive algorithms. This approach leads to a refined measure, M^{abs}, stating the average *absolute size of one non-local routing entry* (i.e., one original subscription that is registered at another broker). This size is calculated by combining practically measured parameters (the reduction in predicates) and findings of our subscription characterization framework (see Section 5.1, page 120), which were shown to hold in practice.

We also apply this absolute measure M^{abs} in combination with subscription covering in conjunctive systems. In this case, those subscriptions that are removed from event routing tables due to covering are included in the average notion of M^{abs} with their actual size of zero. For example, if half of the non-local routing entries are removed from the system by subscription covering (assuming they represent average entries), M^{abs} is reduced to half of its original value.

Reduction in Subscription Routing Table Size. To evaluate the effect of advertisement pruning, we apply a similar high-level measure for subscription routing tables as defined for event routing tables. The memory requirements measure, M^{adv} (*proportional reduction in subscription routing table size*), expresses the proportional change in predicate-advertisement associations. The difference to measure M is that M^{adv} considers the registered advertisements (which can be pruned) instead of the registered subscriptions.

As for subscriptions, our characterization framework from Chapter 5 can be used to describe the absolute memory requirements for advertisements. This framework thus allows us to determine whether a conjunctive or a Boolean solution is generally more space-efficient for given advertisements.

Characteristic Parameters

We require the definition of some characteristic parameters (see Table 8.2 for an overview) to quantify intrinsic attributes of an analyzed test setting, for example, the cover among subscriptions, or the overlap between subscriptions and advertisements.

Proportional Pruning. In some of the experiments, we investigate the behavior of the previously described measures when performing varying numbers of pruning operations. We use the proportional characteristic parameter pru^{prop} (*proportional pruning*) for the number of executed pruning operations. It is always true that $0.0 \leq pru^{prop} \leq 1.0$. For example, $pru^{prop} = 0.0$ describes the un-optimized situation without any pruning, $pru^{prop} = 0.4$ means that 40 percent of all possible pruning operations are performed, and $pru^{prop} = 1.0$ indicates that all possible pruning is executed.

We say that further pruning is impossible if each remaining pruning option removes a complete subscription. These potential pruning operations (i.e., the pruning of the root nodes of subscriptions) were identified as invalid (see Section 6.3, page 155).

Covering Proportion. When comparing subscription pruning to covering, one requires a characteristic parameter that quantifies the amount of cover among subscriptions. For this purpose, we define the *covering proportion*, cov^{prop}, as the number of removed subscriptions due to applying covering divided by the original number of non-local subscriptions within the pub-sub system. Covering is applied for conjunctive subscriptions—the covering proportion thus takes into account converted (i.e., conjunctive) subscriptions.

Generally the larger cov^{prop}, the more covering relationships exist in the system; it always holds that $0.0 \leq cov^{prop} < 1.0$. For example, let us assume that there are 100,000 non-local event routing entries (i.e., subscriptions) in the system without applying covering. When optimizing by subscription covering, the number of non-local routing entries is reduced by 40,000 entries to 60,000 subscriptions. The covering proportion $cov^{prop} = \frac{40{,}000}{100{,}000} = 0.4$.

Large values of cov^{prop} generally occur for highly similar subscriptions, small attribute domain sizes, or few predicates per subscription. Our application-scenario analysis in Chapter 3 did not lead to a typical value for cov^{prop} in the auctioning scenario (we directly derived typical event loads and indirectly applied our results to subscriptions and advertisements).

In our experiments, we create subscriptions with different covering proportions by varying parameter B_{prop}. This changes, for example, the domain sizes of the attributes `Author` and `Title`.

Overlapping Proportion. To quantify the overlap between subscriptions and advertisements, we define the *overlapping proportion*, ovl^{prop}, as a characteristic parameter for the number of subscriptions (proportional to all registered ones) that overlap an advertisement on average. The overlapping proportion is calculated as the average for a large number of advertisements in our experiments (at least 15,000 advertisements).

For example, if 100,000 subscriptions are registered, a value of $ovl^{prop} = 0.2$ means that, on average, 20,000 subscriptions overlap every given advertisement. It always holds that $0.0 \leq ovl^{prop} \leq 1.0$.

The overlapping proportion in a given setting is generally determined by the structure of the registered advertisements and subscriptions. The pruning of subscriptions or advertisements might thus change ovl^{prop}.

Candidate Proportion. To better understand the behavior of the two overlapping efficiency measures (E^{fct} and E^{dec}) when pruning, we need to consider the number of candidates that is evaluated by the system. For this purpose, we define the *candidate proportion*, $cand^{prop}$, as a characteristic parameter for the number of candidates in the Boolean overlapping calculation algorithm.

The candidate proportion is defined as the number of candidates that is evaluated to solve the overlapping function problem after some pruning operations, in proportion to the number of candidates in the un-optimized (i.e., unpruned) setting. Parameter $cand^{prop}$ is always derived by considering at least 50,000 advertisements in the experiments.

For example, if the unpruned setting evaluates 10,000 candidate subscriptions per advertisement on average and after some pruning 15,000 candidate subscriptions need to be evaluated, it holds that $cand^{prop} = \frac{15{,}000}{10{,}000} = 1.5$. As a rule of thumb, an increased number of candidate evaluations, that is, $cand^{prop} > 1.0$, leads to a decreasing overlapping efficiency for the function problem. Note that $cand^{prop} \geq 0.0$.

Number of Brokers. The overall number of brokers in the pub-sub network is given by $|B|$. We vary the number of brokers between $|B| = 5$ and $|B| = 100$.

Note that the network load measure N is given proportional to $|B|$. Thus N does not directly depend on the number of brokers $|B|$.

Number of Registered Subscriptions. In accordance with our subscription characterization framework in Chapter 5, we refer to the number of subscriptions that is registered with a pub-sub system by $|s|$.

8.2 Un-optimized Distributed Filtering

In this section we comparatively analyze the filtering processes in un-optimized system settings (un-optimized with respect to routing, i.e., we do not apply any routing optimizations). These experiments extend the comparative evaluation of our general Boolean algorithm and the counting algorithm. For the counting approach, we compiled a conjunctive version of BoP, which removes the slight overhead of the Boolean filtering approach for conjunctive subscriptions and applies comparable data structures for both solutions. The conjunctive version thus utilizes the plain counting algorithm. As an extension to the setting in Section 5.7.2, the general Boolean algorithm can apply the filtering shortcut optimization in the distributed system.

8.2.1 Filtering in the Distributed Setting

This subsection presents the results of a comparison of the general Boolean filtering algorithm and the counting algorithm in a distributed setting. We expect that the evaluation results for the filtering in individual broker components (as presented in Section 5.7) similarly hold in a distributed setting.

We restrain the pub-sub network to contain five brokers in this set of experiments, that is, $|B| = 5$. We evaluate a line network topology and our results are depicted in Figure 8.1. On the abscissae, we show the number of registered subscriptions $|s|$ up to $|s| = 500,000$. We intentionally chose this maximal number of registered subscriptions to result in index structures that still fit into main memory for the conjunctive counting algorithm. We refer to our work in [BH05b] for results that show the breakdown of the counting approach for larger subscription numbers.

The ordinate shows E, the average filtering time per incoming event message in milliseconds. The results are derived from the combined setting, containing subscriptions of all three subscription classes. We evaluate five predi-

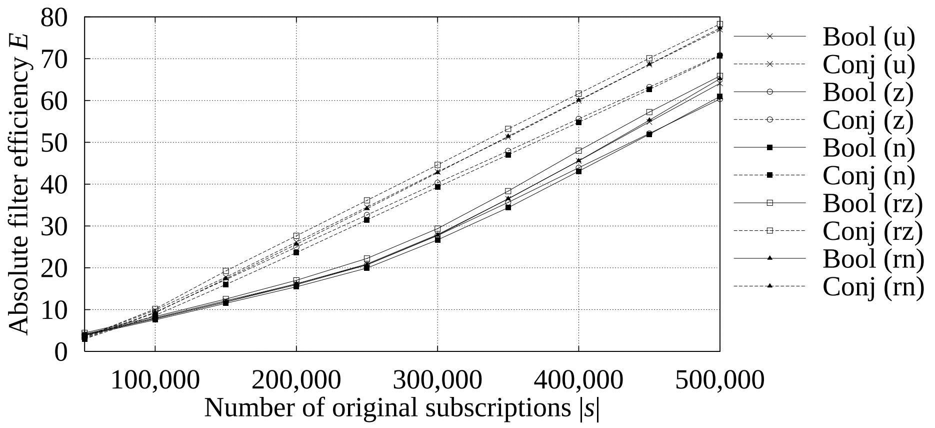


Figure 8.1: Comparison of the filter efficiency E of an un-optimized conjunctive system and an un-optimized Boolean system using a line topology. Predicate distributions are given in brackets.

cate distributions: uniform distribution (u), normal distribution (n), Zipf distribution (z), reversed normal distribution (rn), and reversed Zipf distribution (rz).

The trend for filter efficiency in the distributed setting is similar to their development in central broker components (see Figure 5.5). However, the change in the gradient of the curves is not as significant as in the centralized setting: the effect of distributing event messages within the system accounts for a significant proportion of overall filtering time. The processing required for this task is relatively high compared to the difference of calculating in main memory or in processor cache. Hence the effect of the processor cache in the distributed setting is not as influential as in the centralized setting.

For 500,000 registered subscriptions, the un-optimized Boolean system is between 14 and 17 percent more efficient than the un-optimized conjunctive system for the five tested distributions (compared to approximately 27 percent improvement in the centralized system).

The reason for this difference is that in the distributed system various messages are filtered out in their local broker. These messages do not match any subscriptions within the overall system. Other messages, however, are forwarded and thus evaluated in different brokers. These forwarded messages are relatively "expensive" to evaluate in the general Boolean approach, that is, they lead to a large number of candidate subscriptions. Hence, in the un-optimized distributed system, forwarded event messages do not represent average messages but messages that are "expensive" to evaluate. This property increases the overall filtering time in the general Boolean system compared to the conjunctive version.

Nevertheless, the advantages of a general Boolean system approach are still apparent, as can be seen in Figure 8.1.

8.2.2 Influence of the Filtering Shortcut

We generally expect that the reduction in candidate subscriptions due to the shortcut optimization improves overall filter efficiency. In particular, the more subscriptions are registered, the higher the expected performance increase compared to the un-optimized Boolean approach.

We here show the results for the combined setting, involving all three subscription classes and using the line topology of five brokers. We again analyze five predicate distributions. The results are depicted in Figure 8.2 with an increasing number of registered subscriptions $|s|$ on the abscissa and filter efficiency E on the ordinate.

The results show the changing gradient for both the un-optimized ("Bool") and the optimized ("Short") version of BoP. As expected, the influence of applying the filtering shortcut increases with the number of registered subscriptions, for example, for a uniform predicate distribution, the filtering time per event message E decreases by approximately 10 milliseconds with 500,000 registered subscriptions. Having registered half this number of subscriptions, the difference is approximately 5 milliseconds only.

The reason for this property is clearly the number of candidate subscriptions that do not need to be evaluated when applying the filtering shortcut: the more registered subscriptions, the more candidates exist for a matching message on average. The filtering time decreases significantly if various candidate evaluations are avoided. Therefore, a growing subscription base leads to a greater performance increase when applying the filtering shortcut.

8.2.3 Summary

In this section we comparatively analyzed the un-optimized Boolean filtering algorithm and the un-optimized conjunctive counting algorithm in a distributed setting. The results show that, compared to each other, both algorithms lead to similar results as in the centralized setting (see Section 5.7). Although in a distributed setting the advantage of a general Boolean filtering

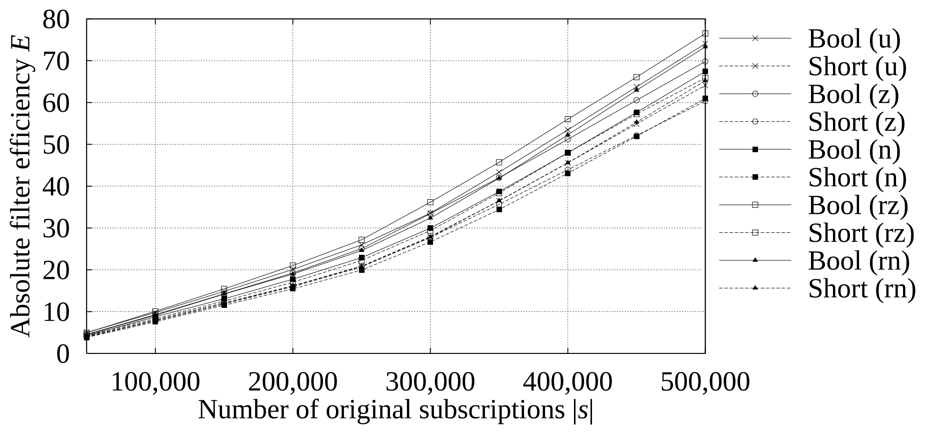


Figure 8.2: Influence of the filtering shortcut on filter efficiency E in a Boolean system using the line topology. Predicate distributions are given in brackets.

algorithm with respect to filter efficiency is less than in a centralized setting, the benefit of a general Boolean approach could still clearly be seen.

Relating these findings to the central hypothesis of this dissertation (Section 1.3, page 6), in this section we have been able to show that an un-optimized distributed Boolean pub-sub approach leads to higher filter efficiency than an un-optimized distributed conjunctive pub-sub approach. Furthermore, the memory requirements of a Boolean approach are less than the memory requirements of a conjunctive approach (according to our findings in Chapter 5). The first part of our hypothesis can thus be transferred to distributed systems.

The second part of this section showed the advantageous influence of the proposed filtering shortcut optimization on the Boolean filtering algorithm. The potential of this optimization increases with a growing number of registered subscriptions.

8.3 General Evaluation of Subscription Pruning

In this section, we analyze the general influence of the subscription pruning routing optimization (see Chapter 6) on a distributed pub-sub system. In particular, we investigate the effect of subscription pruning on filter efficiency E, event routing table size M, and (internal) network load N. In Section 8.3.1 we evaluate subscription pruning for general Boolean subscriptions. Section 8.3.2 considers a conjunctive setting, that is, subscriptions are converted canonically.

Finally, we show the system behavior in the course of subscription pruning when scaling the network size for two extreme topologies in Section 8.3.3. In Section 8.4 we then show the applicability of subscription pruning for different degrees of cover among subscriptions.

8.3.1 Subscription Pruning for Boolean Subscriptions

For this evaluation we use the combined setting that contains subscriptions of all three subscription classes. We show results for a line topology with five broker components, that is, $|B| = 5$. The effect of subscription pruning is largely independent of the chosen network topology and size, as we show in Section 8.3.3. For the experiments in this section, we register 200,000 subscriptions ($|s| = 200,000$) with BoP; other subscription numbers lead to similar results (as described in Section 8.3.2).

Brokers are set up to perform post-pruning in these experiments. Our methodology is to analyze the system, perform some pruning operations in each broker, and analyze the system again. This cycle ends when all possible pruning operations are performed in all brokers. That is, any possible pruning operation in any broker would remove a complete subscription. Adopting this methodology allows us to analyze the system for varying numbers of pruning operations.

Sections 6.4.2 to 6.4.5 introduced four variants to rank pruning operations. We evaluate them subsequently; then we compare these variants to investigate whether they fulfill their individual design goals.

Accuracy-based Pruning

We expect accuracy-based pruning to reduce the memory requirements for event routing tables while only marginally affecting the network load in the system. The pruning of subscriptions generally reduces their complexity. Due to the expected marginal effect of accuracy-based pruning on the network load, the system efficiency should increase in the course of pruning.

In Figure 8.3 we show the influence of accuracy-based pruning on system efficiency, network load, and memory requirements. We show all three of these measures in the figure by directly mapping the reduction in routing table size M (abscissa) onto filter efficiency E (left ordinate) and internal network load N (right ordinate).

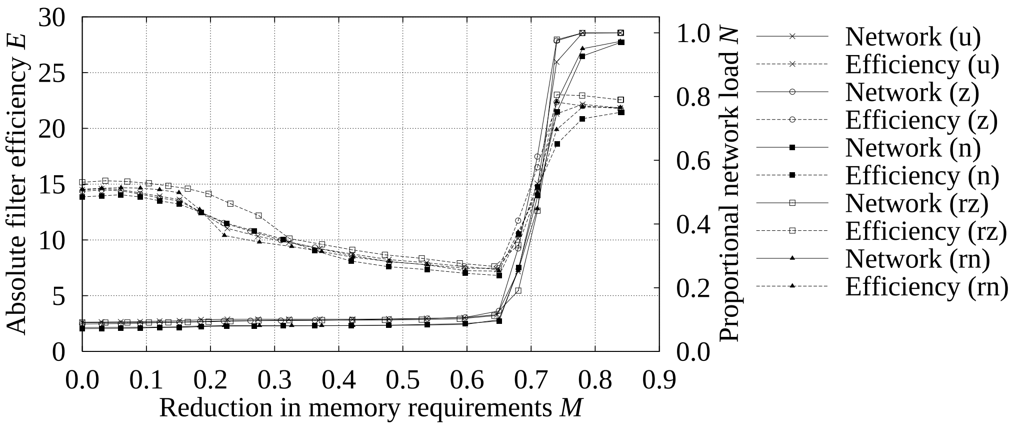


Figure 8.3: Influence of pruning on filter efficiency E (left ordinate) and network load N (right ordinate) using the accuracy-based pruning variant. Predicate distributions are given in brackets.

Accuracy-based pruning shows a relatively stable behavior over the five tested predicate distribution. The resulting graphs are similar for all distributions. Overall, the filter efficiency E is improved from approximately 15 milliseconds per message to 7 milliseconds. That is, the system throughput approximately doubles due to pruning. This result clearly meets our expectation of accuracy-based pruning to improve system efficiency by reducing the complexity of routing entries.

An analysis of the curves in Figure 8.3 shows that filter efficiency E increases up to a certain point on the abscissa, that is, until a certain reduction in routing table size M is reached. After that E starts to degrade (increasing filtering times). We refer to this point of changing system behavior as the *cut-off point*.

One can directly relate this cut-off point to the (increasing) network load N in the system (right ordinate): at the cut-off point, there is a sharp bend in the curves representing the network load N, that is, the number of false positives increases significantly. These false positives are caused by the pruning operations, considerably decreasing the accuracy of pruned subscriptions.

For the accuracy-based variant, the cut-off point occurs after having performed a relatively large number of pruning operations. The routing table size is reduced by approximately 66 percent ($M \approx 0.66$). At the same time, the system throughput more than doubled. This behavior aligns with our expectation that the accuracy-based optimization variant selects pruning operations based on their influence on subscription accuracy.

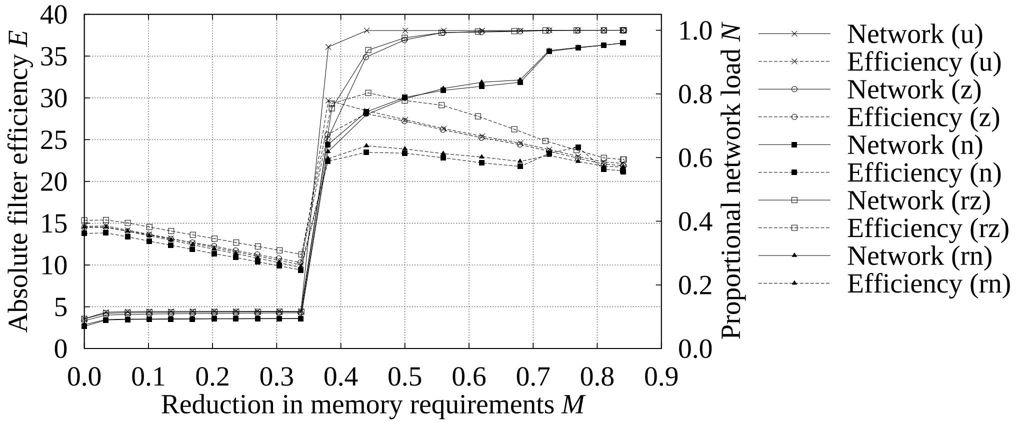


Figure 8.4: Influence of pruning on filter efficiency E (left ordinate) and network load N (right ordinate) using the efficiency-based pruning variant. Predicate distributions are given in brackets.

After the cut-off point, the network load finally reaches its maximal value of $N \approx 1$, that is, each message is forwarded to each broker. At the same time, the system efficiency degrades strongly because of the filtering overhead for false positives. Hence the effect of filtering more messages outweighs the effect of filtering based on less complex routing entries. Before the cut-off point, however, the influence of these effects is reversed (as we expected for accuracy-based pruning).

Efficiency-based Pruning

Efficiency-based pruning was designed with the goal of increasing the system performance. The developed measure for efficiency improvement is based on the minimal number of fulfilled predicates $p_{min}(s)$, and approximates the effect of pruning on system efficiency.

We expect efficiency-based pruning to increase system efficiency E. However, the network load is not considered by efficiency-based pruning. If the pruning operations increase the number of false positives significantly, the overall system efficiency might degrade due to the overhead of filtering these false positives.

The results for efficiency-based pruning are illustrated in Figure 8.4. We use the same mapping of parameters in this figure as before. Up to the cut-off point, there is a steady increase in filter efficiency E: the filtering times decrease from approximately 15 milliseconds per event message (in the unoptimized setting) to 10 milliseconds (at the cut-off point). This result meets

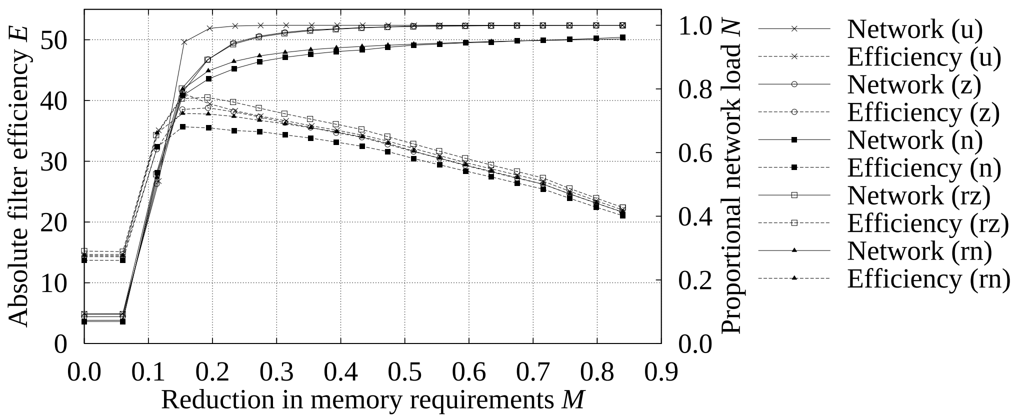


Figure 8.5: Influence of pruning on filter efficiency E (left ordinate) and network load N (right ordinate) using the memory-based pruning variant. Predicate distributions are given in brackets.

our general expectation for efficiency-based pruning.

However, a further decrease in filtering times is counteracted by the strong increase in false positives at the cut-off point. This point already occurs when performing a relatively small number of pruning operations (compared to accuracy-based pruning): approximately 34 percent of the original predicate-subscription associations are removed from the system at this point ($M \approx 0.34$ on the abscissa). Although this early cut-off point does not fully meet our expectations for efficiency-based pruning, the increase in system efficiency for this pruning variant occurs faster than in any other pruning variant.

As in accuracy-based pruning, the increase in false positives at the cut-off point causes degrading filter efficiency. Although the overall performance increases again when pruning beyond this point, BoP does not reach its original event throughput again. The behavior at the cut-off point for different distributions varies slightly, but after all pruning operations the results for all distributions are on a par with each other.

Memory-based Pruning

The target parameter for memory-based subscription pruning is the memory requirements for event routing tables. We thus expect this pruning variant to lead to a stronger reduction in memory requirements than the other pruning variants (see page 252). With respect to network load and system efficiency, memory-based subscription pruning does not consider the effect of pruning operations on false positives. The network load thus might sharply increase at

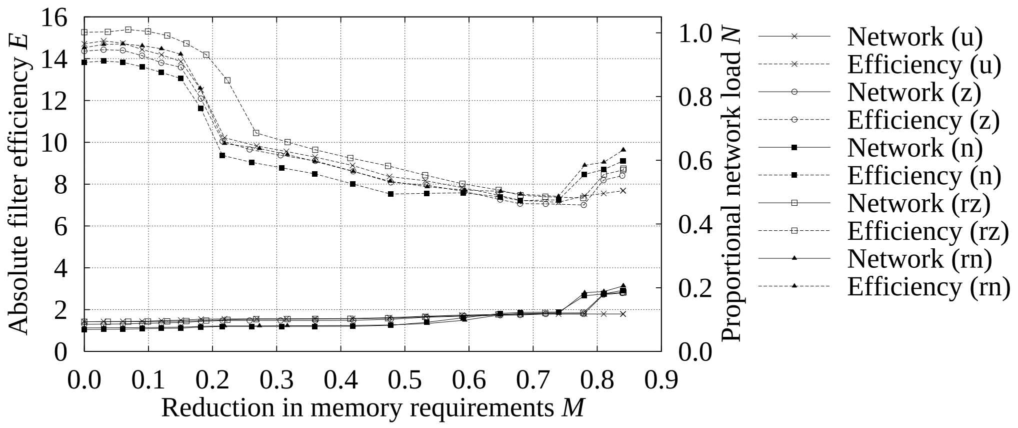


Figure 8.6: Influence of pruning on filter efficiency E (left ordinate) and network load N (right ordinate) using the accuracy and occurrence-based pruning variant. Predicate distributions are given in brackets.

some point during pruning, in turn degrading system efficiency.

Figure 8.5 shows the behavior of memory-based subscription pruning. The cut-off point appears after having reduced the routing table size by only 7 percent ($M \approx 0.07$ on the abscissa). This effect was expected and is due to not taking into account the accuracy of pruned subscriptions. At a reduction of memory requirements by 20 percent ($M \approx 0.2$), nearly all messages are forwarded to all brokers in the network ($N \approx 1$, right ordinate). This alteration of the internal network load N strongly affects system efficiency E (left ordinate), degrading sharply at the cut-off point ($M \approx 0.07$).

However, the system performance again improves when further pruning operations are executed because the maximal possible number of false positives is reached shortly after the cut-off point. Nevertheless, the un-optimized throughput is not reached again. The order of pruning operations does not have a strong influence anymore once the system is nearly flooded with messages: each pruning operation reduces the complexity of routing table entries and thus simplifies the evaluation process for event routing tables, leading to an improving filter efficiency E from $M \approx 0.15$ onwards.

The effects of memory-based pruning are also largely independent of the predicate distributions of subscriptions, as shown in Figure 8.5.

Accuracy and Occurrence-based Pruning

The three previously analyzed pruning variants show a similar overall effect on the system: up to an individual cut-off point, the filter efficiency E increases.

At the cut-off point, the network load N increases sharply, which leads to a degradation in efficiency E. After the cut-off point, the filter performance improves again but never recovers completely.

For accuracy and occurrence-based pruning, however, we expect a different behavior. The cut-off point should only be minor compared to the other measures because of the approach of this pruning variant of increasing the similarity among subscription trees. Hence, the network load N, when performing various pruning operations, should not increase as strongly as in the other variants, in turn reducing its effect on filter efficiency E.

Figure 8.6 shows the system behavior when using accuracy and occurrence-based pruning. There is no major cut-off point in Figure 8.6. We see only a slight increase in network load N for non-uniform predicate distributions at a point where the routing table size is decreased by approximately 75 percent ($M \approx 0.75$ on the abscissa). However, the average proportion of non-local brokers per message N only increases from $N \approx 0.08$ to $N \approx 0.17$, which, as expected, is marginal in comparison to the previous pruning variants. (For the analyzed network size, this is an increase from approximately 0.4 non-local brokers per message to approximately 0.85 non-local brokers per message.)

Due to this minor cut-off point for accuracy and predicate occurrence pruning, the system performance does not degrade as significantly as in the other pruning variants after performing all possible pruning operations. The average time per event message improves from $E \approx 15$ milliseconds in the un-optimized setting through $E \approx 7$ milliseconds as the optimum to $E \approx 9$ milliseconds after executing all pruning operations (averages for different predicate distributions).

The accuracy and occurrence variant thus fulfills its design goal of reducing the number of false positives in comparison to pure accuracy-based pruning. This reduction directly affects the system performance. It does not degrade too much when performing all possible pruning operations.

Again, the results are independent of the predicate distribution in subscriptions.

Routing Table Size

Figure 8.7 compares the influence of the four pruning variants on the routing table size M. We directly plotted the measure for the number of performed pruning operations, pru^{prop}, on the abscissa in this figure. Averages of our re-

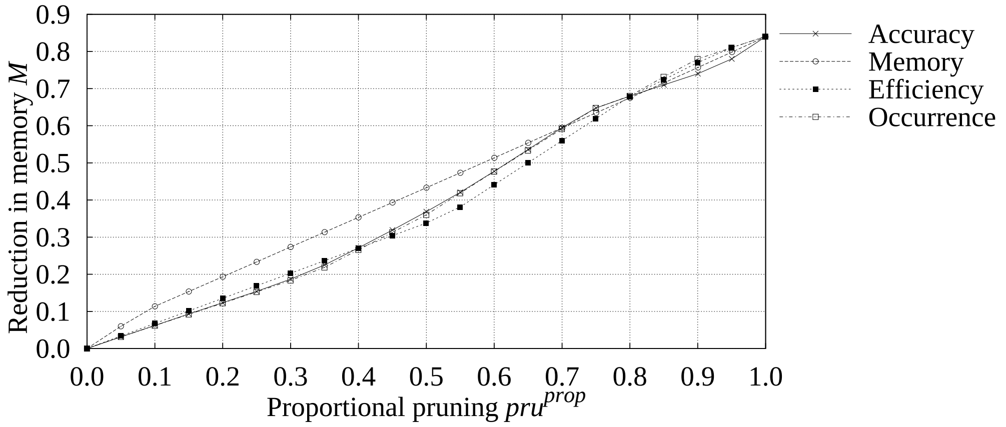


Figure 8.7: Influence of pruning on memory usage M using different pruning variants.

sults for all five of the analyzed predicate distributions are used (the individual results are relatively similar).

We expect the memory-based pruning variant to lead to the fastest reduction in memory requirements. That is, by performing a particular number of pruning operations, memory-based pruning should relinquish more memory resources than the other pruning variants. The maximal reduction in memory requirements, however, might be similar for all variants because comparable parts of subscription trees remain after pruning.

Figure 8.7 reveals that our expectations are fulfilled: Memory-based pruning reduces the routing table size faster than the other pruning variants. Up to approximately 70 percent of all pruning ($pru^{prop} \approx 0.7$ on the abscissa), a comparable number of pruning operations leads to the highest value of M on the ordinate for memory-based pruning. However, after executing all pruning operations the overall reduction of all four pruning variants is comparable: the routing table size is reduced to approximately 16 percent of its original size (i.e., $M \approx 0.84$ on the ordinate).

Network Load

Our expectation with respect to network load N is that accuracy-based, and accuracy and occurrence-based pruning increase the load less than the other two variants. By this statement, we mean that the cut-off points of the two accuracy variants occur after having performed more pruning operations than in the other pruning variants.

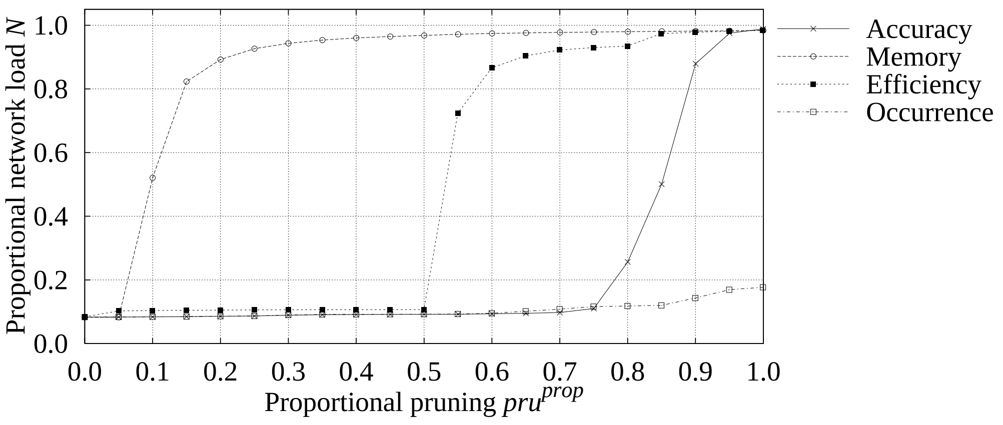


Figure 8.8: Influence of pruning on network load N using different pruning variants.

Figure 8.8 shows a direct comparison of the influence of all four pruning variants on the network load N. The proportional number of pruning operations pru^{prop} is plotted on the abscissa and the proportional network load N is plotted on the ordinate. Again, the averages of all five predicate distributions are used as the basis for the figure.

Figure 8.8 illustrates the different cut-off points for the different pruning variants. Memory-based pruning does not consider false positives at all: the cut-off point (here in terms of proportional pruning) occurs after approximately 5 percent of all pruning operations ($pru^{prop} \approx 0.05$ on the abscissa). The memory-based pruning variant is followed by the efficiency-based variant; its cut-off point is situated at approximately 50 percent of all pruning operations ($pru^{prop} \approx 0.5$). For purely accuracy-based pruning, the cut-off point occurs at $pru^{prop} \approx 0.75$ (abscissa). Accuracy and occurrence-based pruning only shows an insignificant cut-off point at $pru^{prop} \approx 0.85$ because the applied pruning selection variant creates subscriptions that describe similar interests, which prevents a strong increase in false positives.

Hence, our expectations with respect to the increase in network load hold for the two accuracy-based pruning variants.

Filter Efficiency

With respect to filter efficiency, we expect efficiency-based pruning to lead to the fastest improvement in filter efficiency E. However, both accuracy-based variants might result in the best absolute efficiency optimization effect because of the negligence of the increase in network load in efficiency-based pruning.

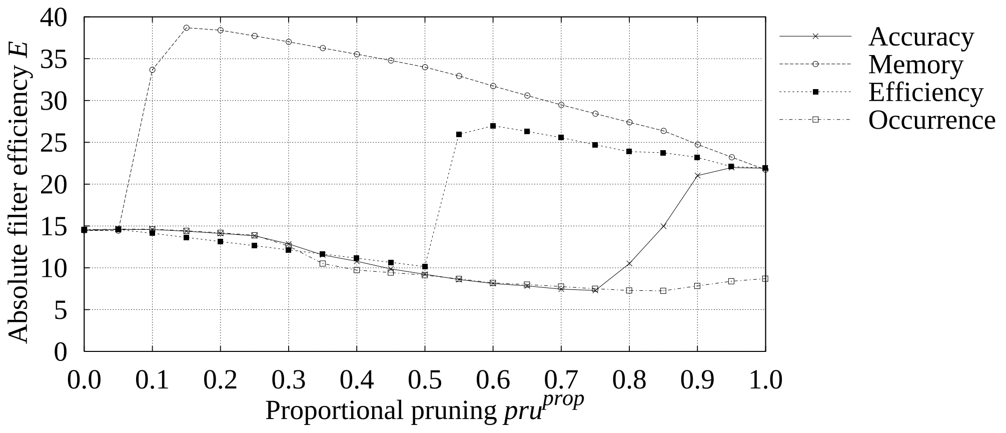


Figure 8.9: Influence of pruning on filter efficiency E using different pruning variants.

Figure 8.9 shows the influence of the four pruning variants on filter efficiency E. The graph is again based on the proportional number of pruning operations pru^{prop} on the abscissa, which is mapped onto the average filtering time per event message E on the ordinate. Average results for the five analyzed predicate distributions are used to obtain the curves.

Fulfilling our expectations, the overall best improvement in filtering time is achieved by the two accuracy-based pruning variants: the system requires approximately 47 percent of the original filtering time per message; the event throughout increases from approximately 67 messages per second to 143 messages per second (improvement from $E \approx 15$ ms to $E \approx 7$ ms).

Although the efficiency-based pruning variant does not result in the best overall filtering performance, it leads to the fastest efficiency improvement by executing the least number of pruning operations (abscissa). This behavior meets our expectations and holds for up to approximately 30 percent of all pruning operations ($pru^{prop} \approx 0.3$). When pruning further, the two accuracy-based variants become more efficient because of the influence of false positives on overall efficiency. Accuracy and occurrence-based pruning can perform all pruning operations without strongly degrading system performance.

8.3.2 Subscription Pruning for Conjunctive Subscriptions

Subscription pruning was developed to optimize general Boolean subscriptions. Due to the focus of subscription pruning on this general subscription class, we

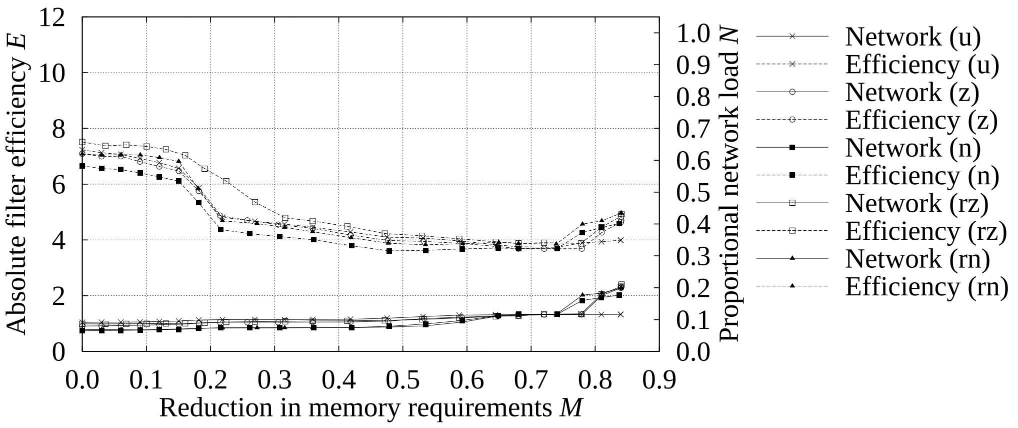


Figure 8.10: Influence of the accuracy and occurrence-based pruning variant on filter efficiency E (left ordinate) and network load N (right ordinate) for Boolean subscriptions. Predicate distributions are given in brackets.

expect that its applicability to restricted conjunctive subscriptions is similar.

In the following experiments, we register 100,000 Boolean subscriptions (which result in 400,000 conjunctive subscriptions after conversion) of all three subscription classes and apply a line topology of five brokers. Results are presented for accuracy and occurrence-based pruning (as they show the best overall results).

In Figure 8.10, we show the influence of pruning in combination with the Boolean filtering algorithm. Figure 8.11 illustrates the results in the conjunctive setting, that is, the counting algorithm and converted subscriptions. In both figures, we map the reduction in routing table size M (abscissae) onto filter efficiency E (left ordinates) and network load N (right ordinates).

Note that the reduction in routing table size M on the abscissa is given proportional to the respective non-local routing entries in both cases: In Figure 8.10, M is based on Boolean subscriptions and in Figure 8.11 on (converted) conjunctive subscriptions. Hence a given reduction in memory requirements, for example, of $M = 0.5$, describes different numbers of removed predicate-subscription associations in both scenarios. We compare the absolute sizes of event routing tables in the course of subscription pruning in Figure 8.12, using the absolute memory requirements measure M^{abs}.

Pruning for Different Subscription Numbers

Before proceeding to the comparison of pruning for Boolean and conjunctive settings, the results of this experiment allow us to relate the effects of subscrip-

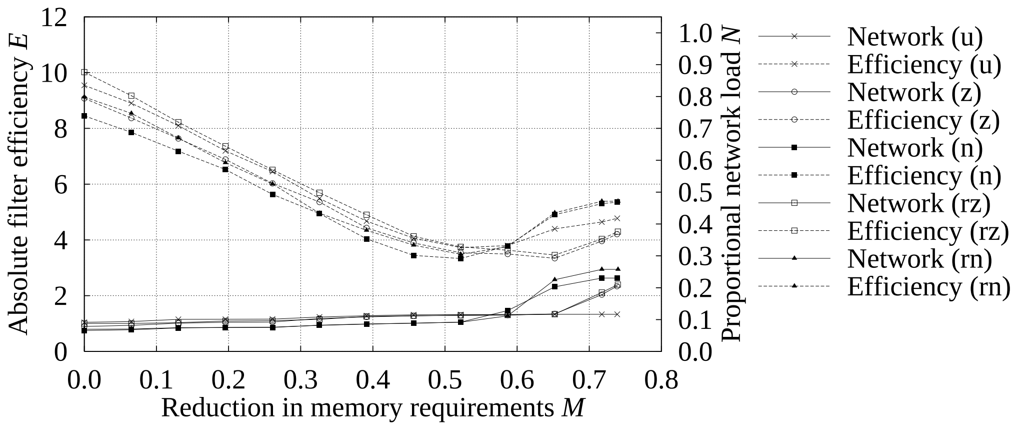


Figure 8.11: Influence of the accuracy and occurrence-based pruning variant on filter efficiency E (left ordinate) and network load N (right ordinate) for converted conjunctive subscriptions. Predicate distributions are given in brackets.

tion pruning for different subscription numbers. We expect that the influence of pruning on network load N, filter efficiency E, and memory requirements M is largely independent of the number of registered subscriptions.

The results in Figure 8.10 represent the system behavior in the same scenario as in Figure 8.3 except for the number of subscriptions ($|s| = 100,000$ in Figure 8.10 and $|s| = 200,000$ in Figure 8.6). The influence of pruning on the network load N (right ordinate) is similar in both settings. For example, for uniform predicate distributions, the load increases from $N \approx 0.09$ to $N \approx 0.12$ in the course of pruning (9 percent of all non-local brokers per message increases to 12 percent of all non-local brokers per message).

The filter efficiency E develops similarly regardless of the number of registered subscriptions. In absolute terms, E depends on the subscription numbers. However, the proportional improvement when performing pruning is comparable: for 200,000 subscriptions, the filter efficiency improves by approximately 52 percent as the maximum and by approximately 40 percent after all pruning. For 100,000 subscriptions, the maximal efficiency improvement is approximately 46 percent; after all pruning operations, E is improved by approximately 36 percent. This general trend was confirmed by further experiments: the proportional efficiency E^{prop} when pruning increases slightly for growing subscription bases. We believe that the reason for this effect is the growing overlap among subscriptions for increasing numbers of registrations.

Finally, the influence of pruning on the routing table size is independent of the number of subscriptions: After all pruning, the size of non-local routing entries is reduced by approximately 85 percent ($M \approx 0.85$). Similarly, the small increase in network load for non-uniform predicate distributions occurs at $M \approx 0.75$ for both sizes of the subscription base. Thus, the optimization potential of subscription pruning is relatively independent of the number of registered subscriptions.

Pruning Conjunctive versus Boolean Subscriptions

We now return to the analysis of subscription pruning for conjunctive settings. Figure 8.10 shows the same original setting as Figure 8.11 except that the former figure is derived from the Boolean version of BoP and the latter figure from the conjunctive version.

We expect that subscription pruning is similarly applicable to both scenarios. The overall possible (proportional) reduction in routing table size M should be higher for the Boolean setting, because at least one predicate remains for each subscription—in the conjunctive case, the number of subscriptions and thus the number of remaining predicates increases (approximately four conjunctive subscriptions are created per original Boolean subscription due to conversion). The filter efficiency E should show similar improvements for the Boolean and the conjunctive version. We also expect a similar behavior with respect to the network load N.

Routing Table Size. When comparing Figures 8.10 and 8.11, one realizes that the maximal reduction in routing table size M is larger for the Boolean setting ($M \approx 0.85$) than it is for the conjunctive setting ($M \approx 0.75$). This behavior meets our expectations and is due to the conversion of subscriptions.

As argued previously, one cannot directly compare the routing table sizes for the conjunctive and the Boolean scenario using the proportional measure M. We thus apply the absolute memory requirements measure M^{abs} for this comparison.

Figure 8.12 plots the proportional number of pruning operations, pru^{prop}, on the abscissa. This measure is mapped onto the average size of one non-local routing entry M^{abs} on the ordinate[6]. We show two curves, one for the Boolean scenario and one for the conjunctive scenario.

[6]The consideration of non-local entries allows us to only incorporate the effect of subscription pruning. Obviously, local entries are much more space-efficient in the Boolean version, see Chapter 5.

The results in Figure 8.12 show that regardless of the number of pruning operations, a Boolean approach to pub-sub implements more space-efficient routing tables. Although the difference between conjunctive and Boolean systems becomes smaller in the course of pruning (the amount of pruning is represented by the values on the abscissa), the Boolean filtering algorithm always requires less memory. Note that the memory requirements M^{abs}, given on the ordinate, are derived from practically measured parameters in combination with our characterization framework from Chapter 5.

Filter Efficiency. Without optimizing by subscription pruning, the Boolean filtering algorithm shows a higher filter efficiency E than the conjunctive approach: the filtering time per event message E is $E \approx 7.0$ milliseconds in the Boolean system in comparison to $E \approx 9.0$ milliseconds in the conjunctive system (taking the averages of the individual results). When performing subscription pruning, the performance of both approaches converges and finally results in a similar amount of time per filtered message of $E \approx 3.8$ ms. This result describes that in the analyzed setting, the proportional efficiency improvement is higher for the conjunctive version of BoP than for the Boolean version.

The reason for this behavior is that the performance of the conjunctive counting approach is independent of actually filtered messages and registered subscriptions. This property leads to linearly decreasing filtering times per message E for the conjunctive system up to the point of increasing network load (the marginal cut-off point at $M \approx 0.55$ on the abscissa). The Boolean algorithm, however, is additionally influenced by the number of candidates that are evaluated but do not match a filtered message. Its efficiency improvement is thus not as steady as in the conjunctive version of BoP. The optimal filtering times of both versions of BoP are comparable (for this pruning variant).

Network Load. Regarding the network load N, subscription pruning in the conjunctive filtering algorithm and the Boolean filtering algorithm leads to similar effects: the network load increases only marginally up to $M \approx 0.75$ for the Boolean setting and up to $M \approx 0.60$ for the conjunctive setting. For uniform predicate distributions, the network load stays at roughly the same level until all pruning operations are executed.

For the Boolean algorithm in combination with other predicate distributions, N increases slightly afterwards. The conjunctive version shows similar

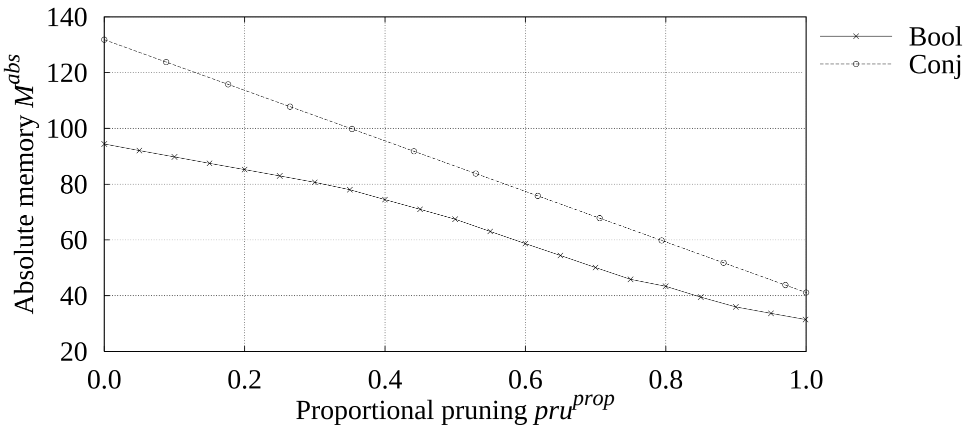


Figure 8.12: Memory requirements for non-local entries in event routing tables M^{abs} when pruning subscriptions in the Boolean and the conjunctive scenario.

increases; however, it leads to a slightly stronger growth in N (but still only marginal in comparison to other pruning variants) for both normal distributions ("n" and "rn" in Figure 8.11). We believe that the reason for this behavior is the effect of conversion on the occurrence of predicates in subscriptions. Because the increase in network load in the conjunctive version is still only marginal, we do not investigate it further at this point.

Altogether, our results show that subscription pruning is effectively applicable to conjunctive subscriptions as well as general Boolean subscriptions.

8.3.3 Subscription Pruning for Different Topologies

In this section we investigate the behavior of subscription pruning for different network sizes and topologies. The accuracy and occurrence pruning variant is chosen for this experiment (as it leads to the best overall results). We expect that the broad optimization effect of subscription pruning occurs regardless of the chosen network topology and scale. The results for different network topologies should only vary slightly. When scaling the network, the increase in network load due to pruning becomes larger in absolute terms; in proportional terms, however, the effect of subscription pruning on the network load is expected to be largely independent of the network size.

We analyze representatives of the two most extreme network topologies in this section. These topologies are the line topology and the star topology. We

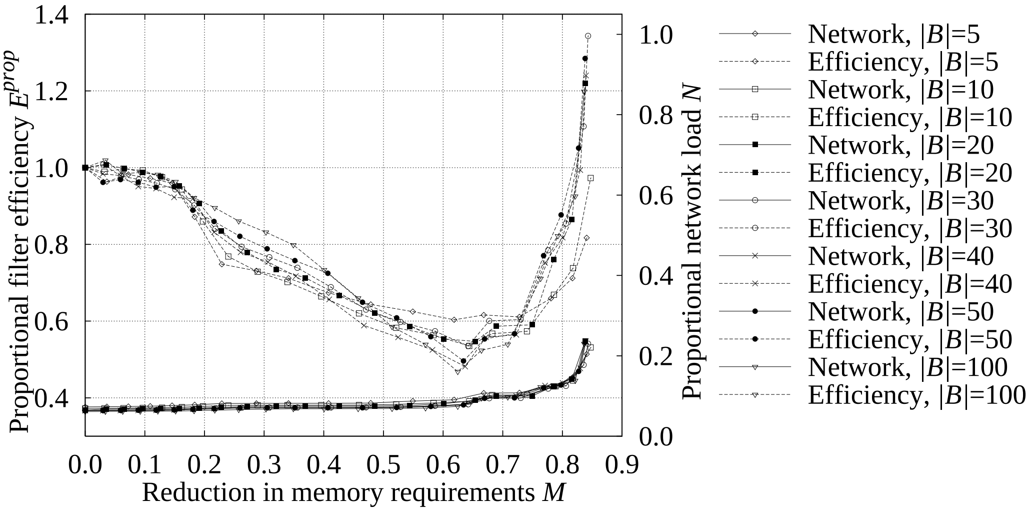


Figure 8.13: Filter efficiency E^{prop} (left ordinate) and network load N (right ordinate) when pruning subscriptions using a line topology for different network scales.

scale the overall network size, that is, the number of brokers, from $|B| = 5$ to $|B| = 100$. Thus, in the line topology the distance from one outmost broker to the other outmost broker is between 4 and 99 hops. In the star topology, between 4 and 99 brokers are connected to the broker in the center of the network[7]. Investigating subscription pruning for these two extreme topologies allows for conclusions about its suitability for general networks, constituting a combination of the properties of the extreme settings.

In these experiments we register 12,500 subscriptions with the overall pub-sub system for all network scales. That is, the number of subscriptions per broker is between 125 (scenario $|B| = 100$) and 2,500 (scenario $|B| = 5$). These subscriptions are uniformly distributed within the network. We run a simulation in this set of experiments, and can thus only evaluate the direct effect of pruning on network load and routing table size. For the filter efficiency, the proportional measure E^{prop} is used for estimation.

Network Load and Routing Table Size

Figure 8.13 illustrates the influence of subscription pruning for seven network scales using the extreme line topology; Figure 8.14 shows the results using a star topology. The abscissae of the figures represent the measure for the

[7]To set the network size in perspective, [Müh02] uses 67 brokers in an experimental evaluation with eight hops as the maximum distance between brokers. [Pie04] uses 10 autonomous systems in a simulation with 100 brokers each.

reduction in routing table size, M. On the left ordinates of both figures, we map the network load N. The figures show the increase in network load within the internal pub-sub system for the different network scales by their individual curves.

Comparing both extreme topologies, the line topology leads to stable results for all analyzed network scales. That is, the proportional number of non-local brokers filtering a message on average, N, remains approximately the same, regardless of the overall broker number and the amount of performed pruning operations (represented by their effect on memory requirements on the abscissa). The reason for this behavior is that the expected value for the number of hops to the local broker of a subscription that is fulfilled by an incoming message remains the same in the line topology, in proportion to the overall network size. Obviously, the absolute number of hops increases with an increasing network scale: for $|B| = 5$, $N \approx 0.06$ (i.e., the un-optimized situation) means that each message is routed via approximately 0.3 non-local brokers; for $|B| = 100$, $N \approx 0.06$ means that each message is routed via approximately 6 non-local brokers.

For a star topology (Figure 8.14) the number of non-local brokers that filter a message on average decreases for an increasing network size. The reason for this effect can be found in the fact that the central broker in a star topology distributes event messages directly to the responsible local broker. Thus the larger the network size, the fewer brokers, on a proportional basis, are involved in the filtering of a message—the number of subscriptions remains the same in our experiments.

The effect of subscription pruning on the network load N is largely independent of the actual network size for both extreme topologies. The magnitude of the cut-off point is similar for both topologies and each of the seven network scales. It occurs at $M \approx 0.65$ and increases the network load by $N \approx 0.1$.

Altogether, our results show that the optimization effect of subscription pruning on network load and routing table size exists regardless of the chosen network topology and scale.

Filter Efficiency

In Figure 8.13, we also show the influence of subscription pruning on filter efficiency for seven network scales using a line topology; Figure 8.14, again, contains the results for the star topology. We directly plotted the measure for

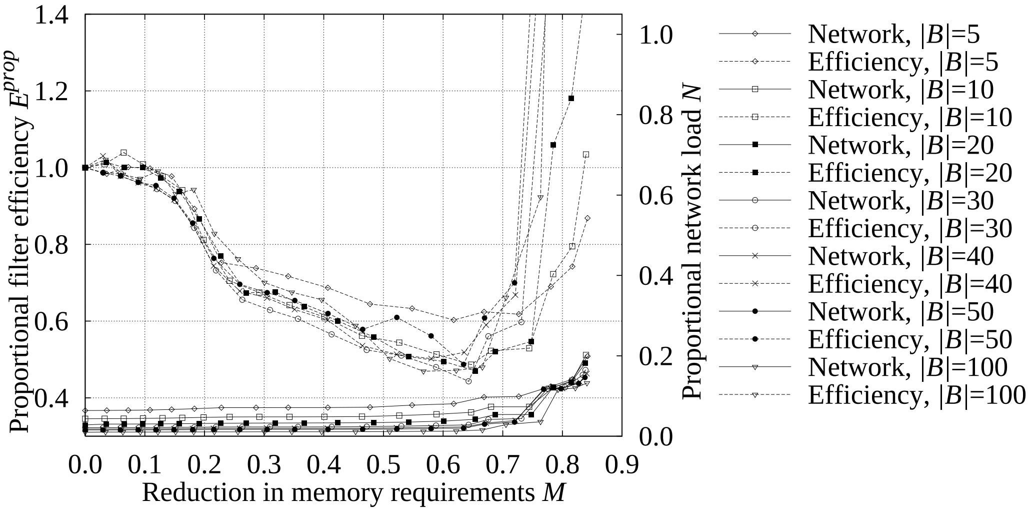


Figure 8.14: Filter efficiency E^{prop} (left ordinate) and network load N (right ordinate) when pruning subscriptions using a star topology for different network scales.

the reduction in routing table size M on the abscissae of the two figures. This measure is mapped onto the proportional filter efficiency E^{prop} on the right ordinates (brokers share physical machines now).

Both figures show that the proportional filter efficiency E^{prop} develops similarly for different network scales. There are also only minor differences between the two analyzed extreme topologies. Generally, the development of filter efficiency is a direct effect of the increase in network load. The best optimization with respect to filter efficiency is between $E^{prop} \approx 0.6$ and $E^{prop} \approx 0.45$. That is, subscription pruning reduces the filtering time per event message between 40 and 55 percent for the analyzed network topologies and scales.

For all seven network scales, the highest filter efficiency is achieved at the previously identified cut-off point ($M \approx 0.65$ on the abscissa), that is, when performing fewer than the maximal possible number of pruning operations. After the cut-off point, the filter efficiency E^{prop} degrades. In particular, in the star topology, E^{prop} degrades more the larger the network becomes. This is due to the load in the central broker in this topology, which experiences the main increase in false positives.

Overall, our experiments show that subscription pruning improves the system efficiency for different network topologies and scales. The optimization effect of subscription pruning with respect to filter efficiency is comparable in the analyzed settings. Although the proportional filter efficiency E^{prop} is an estimation due to the used simulation setting, it gives a good E^{prop} for the system behavior at a larger network scale.

8.3.4 Summary

In this section, we analyzed the influence of subscription pruning on distributed pub-sub systems. One of our main findings is that the four identified pruning variants (accuracy-based, efficiency-based, memory-based, and accuracy and occurrence-based pruning) fulfill their individual design goals. That is, each pruning variant successfully optimizes the system with respect to its target optimization parameter.

Considering the typically applied quality measures in content-based pub-sub systems, both of the accuracy-based pruning variants lead to the overall best system behavior. This result is due to their consideration of the number of false positives that is created in the network as a secondary effect of subscription pruning.

There exist cut-off points for the pruning variants, which indicate up to what point the subscription pruning optimization should be applied in practice. These cut-off points describe what numbers of pruning operations should be executed. When performing more pruning operations than stated by the cut-off point, the network load starts to increase and thus filter efficiency starts to decrease.

We also evaluated the optimization potential of subscription pruning for conjunctive subscriptions. We found that in these settings the pruning optimization leads to similar results as in general Boolean scenarios. Thus subscription pruning can be applied regardless of the subscription structure.

The final part of this section analyzed the influence of the applied broker topology and network size on the effects of subscription pruning. We evaluated the two most extreme topologies. Our findings are that the broad optimization behavior of subscription pruning occurs regardless of the chosen network topology and network scale.

In all of our experiments subscription pruning increased the system efficiency by at least 40 percent and simultaneously reduced the memory requirements for event routing tables by at least 65 percent. The network load was at most 10 percent higher than in the un-optimized setting at this point. These results state the worst case behavior of subscription pruning; most experiments led to far better results.

8.4 Evaluation of Subscription Pruning Under Varying Degrees of Cover

In the previous section we analyzed the intrinsic effects of the subscription pruning routing optimization. What remains to be shown is the applicability of subscription pruning for different covering proportions among subscriptions. We close this gap in Section 8.4.1, allowing us to verify Part 2a of our central hypothesis (page 7).

As an extension to this experiment, Section 8.4.2 shows the system behavior when applying subscription pruning additional to subscription covering. In this experiment, we use the conjunctive version of BoP, which can apply the covering routing optimization.

8.4.1 Subscription Pruning under Varying Degrees of Cover

To verify the suitability of subscription pruning for different degrees of cover, we again use the combined setting with a mixture of subscriptions of all three subscription classes and apply the line network topology using five brokers ($|B| = 5$). We register 100,000 Boolean subscriptions ($|s| = 100,000$), using uniform predicate distributions; the system is then optimized using subscription pruning. The memory requirements are considered using the measure M^{abs} and filter efficiency E. The results we show in the following represent the system behavior at the respective cut-off point.

To correlate the effects of the subscription covering optimization, we also analyze the system behavior under the application of subscription covering. Note that subscription covering is applied in the conjunctive system, whereas subscription pruning is applied in the general Boolean system. Comparing these two settings thus leads to additional conclusions about the behavior of optimized conjunctive and optimized general Boolean systems. According to the properties of the registered subscriptions, 400,000 conjunctive subscriptions are created due to canonical conversion.

We expect that the optimization effect of subscription covering strongly depends on the covering relationships among the registered subscriptions: The more registered subscriptions cover each other, the more optimization potential the covering optimization has. Subscription pruning, on the other hand, is

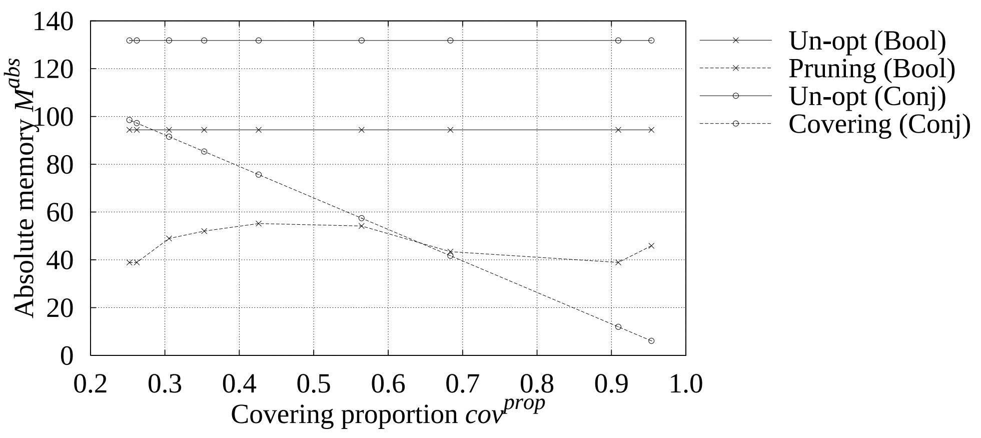


Figure 8.15: Routing table size M^{abs} for different covering proportions (abscissa) using the un-optimized general Boolean algorithm, the general Boolean algorithm with subscription pruning, the un-optimized conjunctive algorithm, and the conjunctive algorithm with subscription covering.

expected to optimize the system regardless of existing covering relationships.

For settings with very high covering proportions cov^{prop}, we expect subscription covering to lead to a stronger reduction in routing table size and a higher filter efficiency than subscription pruning. We argued that these high covering proportions are unrealistic in practice in Section 2.5.2. However, for more realistic scenarios (i.e., with low to moderate covering proportions), subscription pruning should lead to fewer memory requirements and a higher filter efficiency than subscription covering.

In Figure 8.15 and Figure 8.16, we illustrate the experimental results. On the abscissae, we show the covering proportion cov^{prop}, ranging from approximately $cov^{prop} = 0.25$ to $cov^{prop} = 0.95$. Differing covering proportions are derived by varying parameter B_{prop} between $B_{prop} = 0.001$ and $B_{prop} = 100$. On the ordinate in Figure 8.15, we show the memory requirements measure M^{abs}; in Figure 8.16, the efficiency measure E is mapped on the ordinate.

We show four curves in each of the two figures, describing the behavior of the un-optimized general Boolean system ("Un-opt (Bool)"), the behavior of the un-optimized conjunctive system ("Un-opt (Conj)"), the behavior of the general Boolean system under the application of subscription pruning ("Pruning (Bool)"), and the behavior of the conjunctive system under the application of subscription covering ("Covering (Conj)"). Accuracy and occurrence-based pruning is used in this set of experiments. We plotted the individual results for the Boolean system variant according to the covering proportion cov^{prop} of

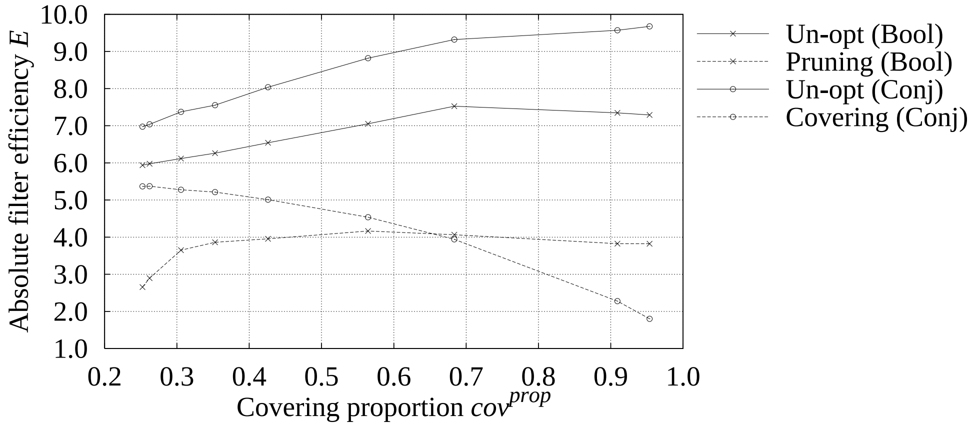


Figure 8.16: Filter efficiency E for different covering proportions (abscissa) using the un-optimized general Boolean algorithm, the general Boolean algorithm with subscription pruning, the un-optimized conjunctive algorithm, and the conjunctive algorithm with subscription covering.

the equivalent conjunctive system setting.

Routing Table Size

Figure 8.15 clearly shows that the optimization effect of subscription pruning ("Pruning (Bool)") does not directly depend on the amount of cover among subscriptions. The memory requirements after the optimization are relatively stable; they generally depend on both the accuracy of the applied pruning variant, and the registered subscriptions that determine the overall potential of pruning. Compared to the un-optimized system ("Un-opt (Bool)"), the memory requirements per non-local routing table entry are reduced from approximately 95 bytes to values between 39 and 54 bytes, that is, a reduction in memory requirements between 44 and 59 percent.

The optimization potential (with respect to memory requirements) of subscription covering ("Covering (Conj)"), on the other hand, depends on the covering proportion cov^{prop}. The fewer subscriptions cover each other (left on the abscissa with small values of cov^{prop}), the higher the memory requirements for event routing tables (higher values of M^{abs} on the ordinate). Thus, if there is almost no cover among subscriptions, hardly any routing table entries are removed by the application of subscription covering. Hence there is a high memory consumption per routing entry in these settings (i.e., large values of M^{abs} on the right ordinate). Evidently, if nearly all subscriptions cover each other, there is a large reduction in memory requirements when applying this optimization (right on the abscissa).

Comparing subscription pruning and subscription covering, pruning leads to more space-efficient routing tables than covering for covering proportions of $cov^{prop} < 0.65$. For higher covering proportions, subscription covering is preferable over subscription pruning. This system behavior meets our expectations. Our application-scenario analysis in Chapter 3 did not allow for a real-world estimation of cov^{prop} for the online auction scenario.

The results show that the memory requirements in un-optimized system settings do not depend on the covering proportion cov^{prop}. Figure 8.15 also depicts the advantage of using an un-optimized Boolean filtering algorithm ("Un-opt (Bool)") over an un-optimized conjunctive filtering algorithm ("Un-opt (Conj)") with respect to memory requirements.

Filter Efficiency

Figure 8.16 shows that filter efficiency E is also not directly influenced by the amount of cover among subscriptions when optimizing by subscription pruning ("Pruning (Bool)"). Near the covering proportion cov^{prop} where the preferable routing optimization changes with respect to memory requirements ($cov^{prop} \approx 0.65$ on the abscissa), the preferable routing optimization with respect to filter efficiency changes as well. Therefore, if there is only a non-extreme amount of cover among subscriptions, subscription pruning leads to higher filter efficiency than subscription covering. Compared to the un-optimized general Boolean system, subscription pruning improves the filter efficiency between 39 percent and 55 percent for the analyzed covering proportions.

When applying subscription covering ("Covering (Conj)"), however, the filter efficiency E directly depends on the amount of cover among subscriptions. Similarly to the effect on the memory requirements, for high covering proportions cov^{prop} (right on the abscissa) subscription covering leads to higher filter efficiency than subscription pruning due to removed routing entries. However, the fewer subscriptions cover each other (i.e., the more left on the abscissa), the less efficient a system becomes applying the subscription covering optimization.

Interestingly, filter efficiency when applying subscription pruning (as well as filter efficiency in the un-optimized system) improves if fewer cover exists among subscriptions. The reason for this behavior is that in case of small covering proportions, only a few published event messages match the registered subscriptions. Thus fewer messages than in the case of high covering proportions are forwarded in the pub-sub network and need to be filtered.

The effect of increasing filter efficiency E with decreasing covering proportions cov^{prop} can also be seen in the un-optimized general Boolean ("Un-opt (Bool)") and the un-optimized conjunctive system ("Un-opt (Conj)"). The subscription covering optimization, on the other hand, cannot effectively remove routing table entries for low covering proportions. This leads to degrading overall filter efficiency with degrading covering proportions.

Comparing filter efficiency E of the un-optimized systems in Figure 8.16 shows the advantage of a general Boolean pub-sub system over a conjunctive pub-sub system without the application of any routing optimizations.

In summary, our experiments show that subscription pruning shows a relatively stable optimization behavior regardless of the amount of cover among subscriptions. As expected, the potential of subscription covering, on the other hand, depends on these relationships. This dependency of subscription covering occurs because this optimization merely exploits the existing covering relationships.

When comparing subscription covering and subscription pruning, covering leads to a better optimization for extreme covering proportions. Subscription pruning optimizes the system for all covering proportions and outperforms covering with respect to space and time efficiency for low to moderate covering proportions.

8.4.2 Simultaneous Covering and Pruning

In Chapter 6, we claimed that subscription pruning can be applied to further optimize pub-sub systems after the full optimization potential of subscription covering is reached. We demonstrate the effect of this dual optimization approach in this section. We expect that the additional optimization effect of subscription pruning depends on the covering proportion: the lower the covering proportion, the higher the additional optimization of subscription pruning with respect to filter efficiency and memory requirements for routing tables.

In this experiment, we apply the same setup as in the comparison of subscription covering and subscription pruning. We use the conjunctive version of BoP, which initially applies the subscription covering optimization.

Figure 8.17 gives an overview of the results. The abscissa contains the cov-

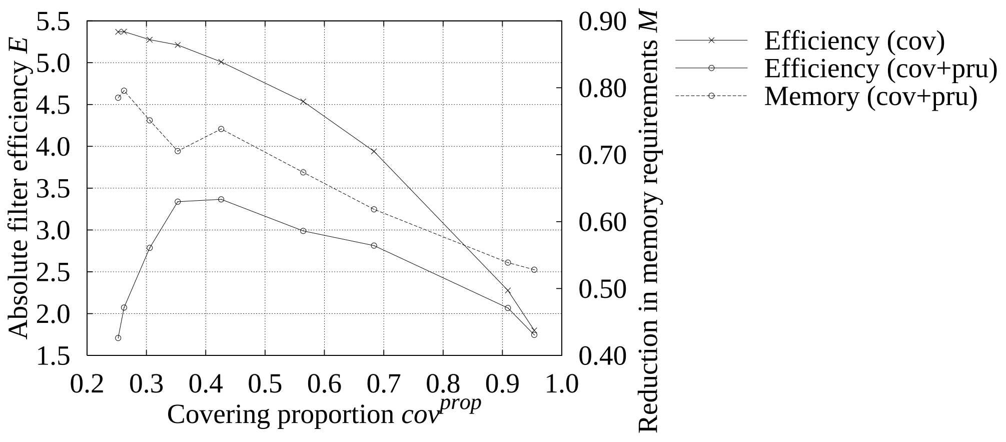


Figure 8.17: Filter efficiency E (left ordinate) and memory requirements M (right ordinate) for different covering proportions cov^{prop} (abscissa) using subscription pruning after subscription covering.

ering proportion cov^{prop} in different scenarios. On the left ordinate, we mapped the filter efficiency E. Two of the curves in the figure use this ordinate—one curve illustrates E after applying subscription covering, another curve illustrates E when additionally applying subscription pruning after subscription covering. The right ordinate in Figure 8.17 contains the reduction in memory requirements M that is achieved by subscription pruning. As a basis for the proportional measure M, the memory requirements after subscription covering are taken.

Filter Efficiency

The fewer covering relationships exist among subscriptions, the more filter efficiency E degrades when only applying subscription covering ("cov" in Figure 8.17). However, subscription pruning is independent of the amount of cover among subscriptions. As can be seen in Figure 8.17 and meeting our expectations, even if the system is fully optimized by subscription covering, there exists a large potential to subsequently optimize by subscription pruning ("cov+pru").

The additional effect of subscription pruning on filter efficiency E is given by the difference between the two curves for E in Figure 8.17 ("cov" and "cov+pru"). For all covering proportions cov^{prop} (values on the abscissa), the filter efficiency E (on the left ordinate) is further improved by subscription pruning. In proportional terms, the efficiency improvement is between 4 and 68 percent, depending on the covering proportion. Generally, the lower the covering proportion cov^{prop}, the higher the additional optimization potential of subscription pruning (values further left on the abscissa). This property is caused by the insufficiency of subscription covering for situations with non-extreme covering proportions. This problem of subscription covering was identified as one of its major shortcomings in practice in Section 2.5.2 (page 50).

The overall development of filter efficiency E in Figure 8.17 is similar to the results when only optimizing by subscription pruning. The best optimization results are achieved for small covering proportions. In the joint optimization high covering proportions lead to a filter efficiency that is higher than in the case of only optimizing by subscription pruning. This result is due to the advantageous property of subscription covering to remove entries in event routing tables, which generally improves system efficiency.

Routing Table Size

The third curve in Figure 8.17 shows the reduction in routing table size M when optimizing by subscription pruning (after optimizing by subscription covering). The results show that the additional optimization potential of subscription pruning (with respect to memory requirements) increases with decreasing covering proportions cov^{prop} (left on the abscissa).

The reason for this property is the inapplicability of subscription covering for settings with low covering proportions cov^{prop}. For the joint optimization, the results show that subscription pruning counterbalances the insufficiency of subscription covering in settings with non-extreme covering proportions (on the left to the middle of the abscissa).

However, even for extreme covering proportions (right on the abscissa), subscription pruning reduces the memory requirements of the system significantly. This advantageous behavior is due to the orthogonal optimization dimensions of subscription covering and subscription pruning—subscription pruning is specifically designed to work independently of the amount of cover among subscriptions. Hence subscription pruning fully fulfills our expectations.

8.4.3 Summary

This section analyzed the applicability of subscription pruning under varying covering proportions. We also evaluated the optimization effect of simultaneously utilizing subscription pruning and subscription covering.

The results validate that the optimization potential of subscription covering strongly depends on the amount of cover among subscriptions, as argued in Section 2.5.2 (page 50). The fewer covering relationships exist among subscriptions, the less is the optimization effect of this event routing optimization. We designed subscription pruning, on the other hand, to be applicable regardless of the cover among subscriptions. Our experiments confirmed this general applicability of subscription pruning and revealed that pruning shows relatively stable results for different covering proportions.

Relating our experimental findings of this section and Section 8.3 to the central hypothesis of this dissertation, our results prove Part 2a of this hypothesis:

> *Subscription pruning increases system efficiency and decreases routing table size, independently of the existing covering relationships.*

Furthermore, our experiments show that for non-extreme covering proportions the filter efficiency of a general Boolean system in combination with subscription pruning is higher than the filter efficiency of a conjunctive system in combination with subscription covering. We derived similar results with respect to the memory requirements for event routing tables: for non-extreme covering proportions, subscription pruning reduces the sizes of these routing tables more than subscription covering does. For settings with extreme covering proportions, however, subscription covering is the preferable routing optimization. We hardly expect such high covering proportions in practice.

In a second experiment we demonstrated that subscription pruning can be successfully applied in addition to subscription covering. The results show that even if the covering optimization reaches its full potential, subscription pruning further improves the system behavior with respect to filter efficiency and routing table size.

8.5 Calculation of the Overlapping Relationships

Having previously evaluated settings without the use of advertisements, we now shift our viewpoint to advertisement-based pub-sub systems. As a first step, we analyze our algorithm to calculate the overlapping relationships between general Boolean subscriptions and advertisements. We then compare this calculation approach to a conjunctive solution for the overlapping calculation. As a second step in the analysis involving advertisements, we evaluate our advertisement-based optimization, advertisement pruning.

In the experiments in this section we register an increasing number of advertisements with the pub-sub system, ranging from 20,000 to 500,000, and evaluate the efficiency of the overlapping calculation. When analyzing the conjunctive overlapping calculation method, the registered advertisements are converted to conjunctive forms. This process leads to 2.5 conjunctive advertisements per original Boolean advertisement on average.

We evaluate uniform predicate distributions for both subscriptions and advertisements. The results we analyze are derived from the combined setting containing advertisements of all eight advertisement classes and subscriptions of all three subscription classes. The presented results are averages for 15,000 original general Boolean subscriptions. For the conjunctive calculation algorithm, these subscriptions are converted canonically (leading to 60,000 conjunctive subscriptions).

8.5.1 General Boolean Approach

We identified earlier that pub-sub systems benefit when distinguishing between the overlapping decision problem and the overlapping function problem: the function problem determines all overlapping advertisements for a subscription, whereas the decision problem determines whether at least one overlapping advertisement exists.

In this section we analyze the efficiency of our Boolean algorithm to calculate the decision problem, E^{dec}, and the function problem, E^{fct}, for an increasing number of registered advertisements. We expect linearly increasing calculation times for both overlapping function problem and overlapping decision problem. The decision problem is expected to be much more efficient to calculate than the function problem (see Section 7.2.4).

In Figure 8.18, we illustrate the number of registered advertisements on the abscissa; on the ordinate we map the overlapping calculation efficiency. We show individual curves for both overlapping problems, one curve for the efficiency of the overlapping function problem E^{fct} and one for the efficiency of the overlapping decision problem E^{dec}. We also show the behavior of the conjunctive calculation algorithm, and compare both approaches in Section 8.5.2.

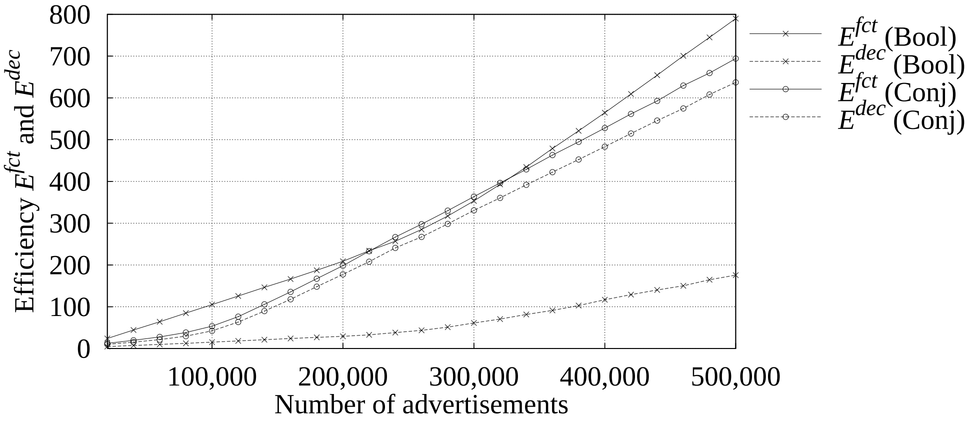


Figure 8.18: Overlapping calculation efficiency in the Boolean and the conjunctive setting. We show the calculation efficiency for the function problem E^{fct} and the decision problem E^{dec} for the Boolean and the conjunctive calculation approach, respectively.

Similar to the behavior of the Boolean filtering algorithm, the curves representing the efficiency of the Boolean overlapping calculation algorithm in Figure 8.18 show a changing gradient with a growing number of registered advertisements (abscissa). As for the filtering algorithm, the reason for this behavior is the incrementing of counters per subscription in the hit vector. In the overlapping calculation algorithm, these counters represent the number of disjoint predicates per subscription (in the filtering algorithm, it is the number of fulfilled predicates).

The point of changing gradients occurs at approximately 200,000 registered advertisements (on the abscissa). This number of registered advertisements is similar to the number of registered subscriptions in the experiments for the filtering algorithm. Because the processor cache advantageously influences the incrementing of predicate counters in both algorithms, the point of changing gradient occurs at a similar number of subscriptions or advertisements. As for the filtering algorithm, the calculation efficiency for the overlapping algorithm is in fact linear. The initial small gradient is due to the influence of the processor cache.

The overlapping decision problem is solved as a shortcut to the function problem in the Boolean algorithm. Thus the decision problem is more efficient to calculate than the function problem, meeting our expectations. A comparison of the calculation efficiency measures for the decision problem E^{dec} and the function problem E^{fct} in Figure 8.18 reveals that the overlapping decision problem is solved in approximately one fifth of the time of the function problem. This proportional difference between both problems is stable for all numbers of registered advertisements.

8.5.2 Comparison to Conjunctive Solution

In Figure 8.18, we plotted the calculation efficiency for the overlapping decision problem E^{dec} and the overlapping function problem E^{fct} for the conjunctive calculation approach. We expect that the difference in calculation time between function and decision problem is marginal in the conjunctive algorithm. This is because the main computational load in the conjunctive approach is to increment the counters for the numbers of disjoint predicates per subscription. This load occurs for both decision and function problem. With respect to the comparison of conjunctive and general Boolean approach, we expect the overlapping decision problem to be much more efficient to calculate in the Boolean algorithm. For the function problem, both solutions should be on a par with each other.

Figure 8.18 shows that there is only a minor difference in computation time between the overlapping decision and the overlapping function problem in a conjunctive system. The time to solve the decision problem is between approximately 80 and 92 percent of the computation time for the function problem. For the Boolean algorithm, the decision problem could be solved in 20 percent of the time of the function problem.

The behavior of a conjunctive calculation algorithm is, as expected, due to its small potential to avoid computations in the decision problem: the conjunctive algorithm always needs to increment all counters for disjoint predicates in the hit vector. If a subscription s leads to zero disjoint predicates, s constitutes an overlap. The counting of disjoint predicates in the conjunctive algorithm, however, is more costly than in the Boolean algorithm because advertisements need to be converted, that is, the overall number of predicate-advertisement associations and thus the number of increment operations increases.

Furthermore, Boolean subscriptions need to be converted by a conjunctive system. For each of the equivalent conjunctive subscriptions, the algorithm then needs to solve the overlapping decision problem[8]. Thus a conjunctive

[8]Conjunctive algorithms cannot optimize the calculation process because the information that several conjunctive subscriptions are created from the same Boolean subscription is lost.

algorithm needs to perform most of its computations for both overlapping decision and overlapping function problem.

Considering the development of the curves for the conjunctive algorithm, there again exists a point of changing gradients. For the conjunctive solution, the change in gradient occurs at approximately 80,000 advertisements (abscissa). As in the filtering algorithm, this difference results from the effect of canonical conversion. This conversion increases the number of registered advertisements by approximately 2.5. Hence, the changing gradient is shifted left on the abscissa by a factor of 2.5.

Comparing Boolean and conjunctive overlapping calculation, the Boolean algorithm solves the decision problem more efficiently than the conjunctive algorithm. For 500,000 registered advertisements, the Boolean solution requires approximately 28 percent of the computation time of the conjunctive solution in the same setting. The overlapping decision problem needs to be solved when subscriptions are registered with a pub-sub system. It is crucial to be solved efficiently. Our Boolean algorithm is the preferable choice in this respect.

The overlapping function problem, on the other hand, is computed more efficiently by the conjunctive algorithm. The reason for this is the large number of candidate subscriptions that need to be evaluated in the Boolean algorithm. For 500,000 registered subscriptions, the conjunctive approach requires 12 percent less time than our Boolean approach (i.e., 88 percent of its computation time).

Considering the final gradients in Figure 8.18, one might even expect this difference between the algorithms to increase for growing advertisement bases. However, by applying the results from our characterization framework in Chapter 5, we derived that conjunctive overlapping calculation approaches require more memory than Boolean overlapping calculation approaches. Thus a conjunctive algorithm cannot be applied for large advertisement bases. In this case, our Boolean solution also becomes the preferable choice for the function problem.

8.5.3 Summary

In this section we compared the efficiency of our Boolean overlapping calculation approach to a conjunctive solution.

We found that a conjunctive calculation algorithm solves the overlapping function problem slightly more efficiently than our Boolean algorithm (for example, 13 percent less time in the conjunctive algorithm for 500,000 registered advertisements). However, the Boolean solution requires less memory than the conjunctive solution. Our Boolean approach is thus applicable in scenarios with high advertisement numbers that cannot be processed by a conjunctive approach.

The overlapping decision problem, on the other hand, shows much better time efficiency properties when using our Boolean algorithm (e.g., 72 percent less time in the Boolean algorithm for 500,000 registered advertisements). This result is particularly promising because the overlapping decision problem needs to be solved when subscriptions get registered; it thus requires an efficient solution, as provided by our general Boolean approach.

8.6 Evaluation of Advertisement Pruning

Having analyzed the overlapping calculation in advertisement-based pub-sub systems, we evaluate our advertisement-based routing optimization, advertisement pruning. We analyze the general influence of advertisement pruning on a pub-sub system, and investigate the effects of applying subscription pruning and advertisement pruning simultaneously in a system.

We only analyze the system behavior in individual broker components. The measurements we present are derived from a scenario with 100,000 subscriptions of all three subscription classes and 50,000 advertisements of all eight advertisement classes.

The experimental methodology is similar to the methodology we applied for subscription pruning: initially, we solve both overlapping problems (function and decision) for all given advertisements based on the registered subscriptions (and take the measurements). Then we perform some advertisement pruning operations, and again solve the overlapping problems. This cycle runs until all possible advertisement pruning operations are performed. In the second experiment in Section 8.6.2, we prune the registered subscriptions before the described cycle starts. The number of performed subscription pruning operations is varied in this second experiment.

Within the following evaluation, we analyze the efficiency to solve the overlapping function problem E^{fct}, the efficiency to solve the overlapping decision problem E^{dec}, the subscription routing table size M^{adv}, the event routing table size M, the overlapping proportion ovl^{prop}, and the candidate proportion

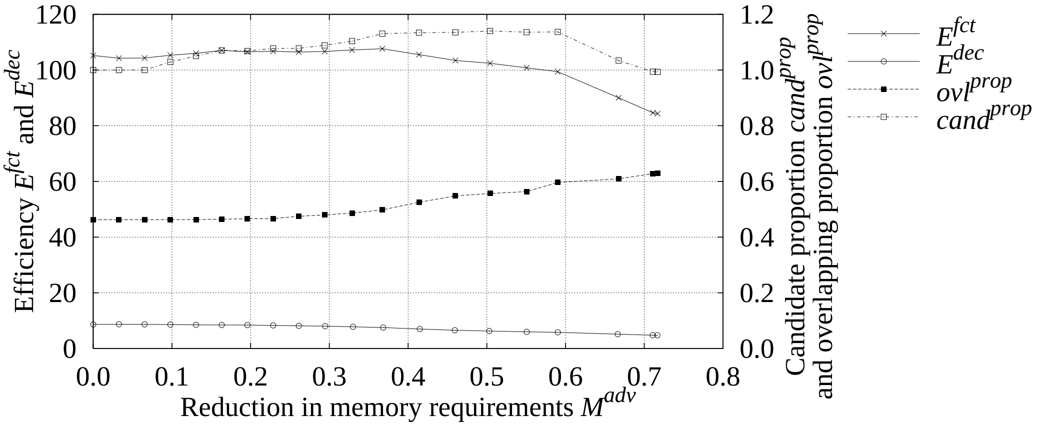


Figure 8.19: Influence of advertisement pruning on routing table size M^{adv}(abscissa), time efficiency for the overlapping calculation E^{fct} and E^{dec} (left ordinate), and overlapping proportion ovl^{prop} and candidate proportion $cand^{prop}$ (right ordinate).

$cand^{prop}$. Ultimately the experiment in the following subsection proves the final part (Part 2b) of our central hypothesis (page 7).

8.6.1 Pure Advertisement Pruning

On the abscissa of Figure 8.19, we plotted the measure for the reduction in the size of subscription routing tables, M^{adv}. This reduction is the primary effect when performing advertisement pruning. On the left ordinate of Figure 8.19, we map the measure for the time efficiency to solve the overlapping function problem E^{fct} and the overlapping decision problem E^{dec}. The right ordinate of the figure contains the overlapping proportion ovl^{prop} and the candidate proportion $cand^{prop}$.

In Figure 8.19, we show four curves, each representing one of the described parameters. Average results are plotted for all five predicate distributions. The results for the individual distributions are similar and do not lead to new insights into the effect of advertisement pruning on pub-sub systems. In each of the individual settings that is incorporated into the average, both subscriptions and advertisements are created using one of the five used predicate distributions.

We expect that advertisement pruning reduces the memory requirements for subscription routing tables. At the same time, the overlap between subscriptions and advertisements should increase slightly as a secondary effect of advertisement pruning. Moreover, we expect that the efficiency for the calculation of the overlapping task increases in the course of advertisement pruning.

Routing Table Size

After all possible advertisement pruning operations are performed, approximately 72 percent of the existing predicate-advertisement associations are removed from the system ($M^{adv} \approx 0.72$ on the abscissa in Figure 8.19). This effect of advertisement pruning constitutes a strong reduction in subscription routing table size and fully meets our expectations.

The maximal reduction in the size of subscription routing tables is less than the maximal reduction in the size of event routing tables when applying subscription pruning. The reason for this effect is the structure of advertisements. For example, for Advertisement Classes 6 to 8 valid advertisement pruning operations always result in more than one remaining predicate per advertisement. Thus the maximal possible reduction in memory requirements for advertisement pruning is less than for subscription pruning, which can always result in only one remaining predicate for Subscription Classes 1 to 3.

Efficiency

In the course of advertisement pruning, the efficiency for calculating both the overlapping function and overlapping decision problem improves. Without any advertisement pruning ($M^{adv} = 0$ on the abscissa in Figure 8.19), the function problem can be solved in $E^{fct} \approx 105$ milliseconds and the decision problem in $E^{dec} \approx 9$ milliseconds. After all possible pruning operations, the efficiency of calculating the function problem is improved to $E^{fct} \approx 85$ milliseconds and the efficiency of calculating the decision problem is improved to $E^{dec} \approx 5$ milliseconds. In the course of advertisement pruning these problems can be solved in approximately 80 and 44 percent of their original time, respectively.

The algorithm to calculate the overlapping relationships is influenced by various factors, for example, the number of predicates in the system. When applying advertisement pruning, this number of predicates decreases, generally improving the computation efficiency. For the overlapping function problem, another important influence on the efficiency of its calculation is the number of candidate subscriptions, represented by $cand^{prop}$ on the right ordinate in Figure 8.19.

The behavior of $cand^{prop}$ and M^{adv} (abscissa) directly relates to the efficiency of solving the function problem, E^{fct} (on the left ordinate in Figure 8.19). The more predicates are removed from the system, the more efficient the algorithm for the function problem becomes (there are fewer disjoint predicates). However, the increase in candidate subscriptions, as a secondary effect of advertisement pruning, counteracts this advantageous effect.

Therefore, E^{fct} approximately stays the same up to $M^{adv} \approx 0.4$ (fewer predicates but more candidates). Between $M^{adv} \approx 0.4$ and $M^{adv} \approx 0.6$, the candidate proportion $cand^{prop}$ does not increase anymore. So the advantageous effect of reducing the number of predicates is directly reflected in the improvement of E^{fct}. For $M^{adv} > 0.6$, the number of candidates decreases[9] (even under its original value). Hence both the reduced number of predicates and the reduced number of candidates advantageously affect the efficiency E^{fct}.

For the decision problem, the main influence on efficiency E^{dec} is the reduction in predicate numbers, always occurring in the course of advertisement pruning. Overlapping candidates affect the computation in the decision problem in the following way: the earlier a candidate constitutes an overlap, the more efficient the calculation algorithm becomes. Thus, increasing candidate numbers might improve E^{dec}. The extreme is that all subscriptions overlap a given advertisement, which requires only one candidate evaluation for the whole computation regardless of the order in which candidates are evaluated.

Amount of Overlap

We expected advertisement pruning to increase the overlap between subscriptions and advertisements as little as possible. The results in Figure 8.19 indicate that this expectation holds in practice.

Without advertisement pruning, it holds $ovl^{prop} \approx 0.46$. After all possible pruning operations, this value increases to $ovl^{prop} \approx 0.63$, a proportional increase of 37 percent. At the same time, the memory requirements for subscription routing tables are reduced by 72 percent (abscissa in Figure 8.19). As we show in Section 8.6.2, this is a much smaller increase in overlap than by applying an optimization that is not tailored to advertisements, such as

[9]Note that this decrease is due to the counting of the number of candidate evaluations in our experiments. If a subscription s is a candidate for several options that are described by an advertisement, $cand^{prop}$ incorporates how many times s is evaluated. Hence, if the first evaluation reveals s as an overlap, further evaluations do not take place and are thus not counted in $cand^{prop}$.

subscription pruning.

Using our cut-off point notion from subscription pruning, one could identify a minor cut-off point at $M^{adv} \approx 0.37$ (increasing the overlap from $ovl^{prop} \approx 0.46$ to $ovl^{prop} \approx 0.5$ in Figure 8.19) or $M^{adv} \approx 0.55$ (increasing the overlap from $ovl^{prop} \approx 0.46$ to $ovl^{prop} \approx 0.56$). Thus, drawing similar conclusions as for subscription pruning, advertisement pruning might only be performed up to this cut-off point.

However, the decision of when to stop advertisement pruning is more complex than merely considering the development of ovl^{prop}. This is because an increase in overlap due to advertisement pruning, on the one hand, means an advantageous decrease in the size of subscription routing tables, but, on the other hand, it means a disadvantageous increase in the size of event routing tables. Finding an optimal balance between these two effects of advertisement pruning is beyond the scope of these experiments.

8.6.2 Combining Advertisement Pruning and Subscription Pruning

So far, our evaluation has considered the application of subscription pruning and advertisement pruning as two independent routing optimizations. In this section we analyze the system behavior when applying subscription pruning and advertisement pruning at the same time.

Similarly to the analysis of advertisement pruning, this experiment considers the efficiency of calculating the overlapping function problem E^{fct}, the efficiency of calculating the overlapping decision problem E^{dec}, the memory requirements for subscription routing tables M^{adv}, the overlapping proportion ovl^{prop}, and the candidate proportion $cand^{prop}$ (here proportional to the unoptimized setting with respect to both subscription pruning and advertisement pruning). We then analyze the effect of subscription pruning on the size of event routing tables, M.

We initially register all subscriptions, perform some subscription pruning operations, and finally optimize the system by advertisement pruning. We expect that subscription pruning leads to a much stronger increase in overlap than advertisement pruning when reducing the memory requirements for routing tables by a similar proportion.

We plotted the results in Figures 8.20 to 8.23. In all four figures, we illustrate the memory requirements for subscription routing tables, M^{adv}, on

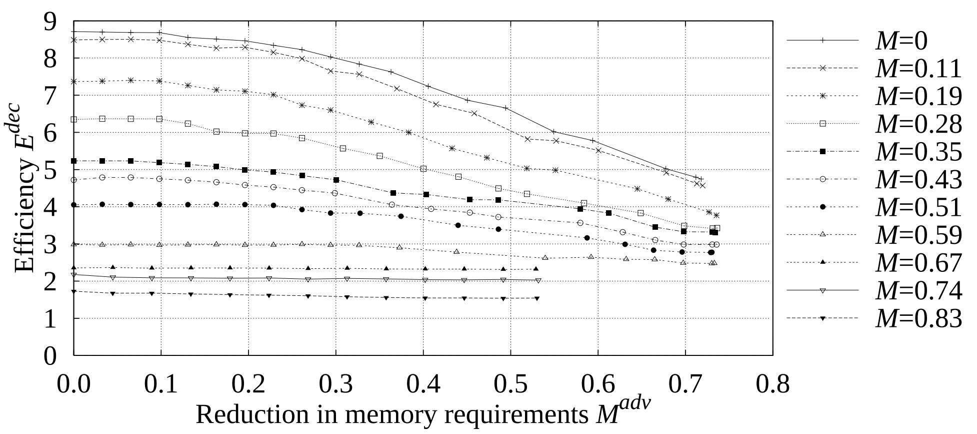


Figure 8.20: Influence of simultaneous subscription and advertisement pruning on the efficiency of the overlapping calculation decision problem E^{dec}.

the abscissa. This reduction is an effect of advertisement pruning. On the ordinate, we show the efficiency of calculating the overlapping decision problem E^{dec} in Figure 8.20, the efficiency of calculating the overlapping function problem E^{fct} in Figure 8.21, the overlapping proportion ovl^{prop} in Figure 8.22, and the candidate proportion $cand^{prop}$ in Figure 8.23.

Each of the figures contains different curves. Every curve is characterized by a certain reduction in event routing table size M. These curves thus represent the system behavior after executing a varying number of subscription pruning operations (accuracy-based pruning). The minimal value of $M = 0$ represents the situation without any subscription pruning; the maximal value of $M = 0.83$ describes the fact that all possible subscription pruning operations are performed.

Routing Table Size

As can be observed in Figures 8.20 to 8.23, the number of performed subscription pruning operations (i.e., the different curves in the individual figures) affects the potential of advertisement pruning to reduce the memory usage of subscription routing tables M^{adv} (abscissa). Without executing any subscription pruning (curves with $M = 0$), advertisement pruning reaches a maximal reduction in memory requirements of $M^{adv} \approx 0.72$. In the course of subscription pruning (other curves), this maximal reduction in subscription routing table size slightly increases to $M^{adv} \approx 0.74$; finally, it decreases to $M^{adv} \approx 0.53$ when all possible subscription pruning operations are performed (curves for $M = 0.83$).

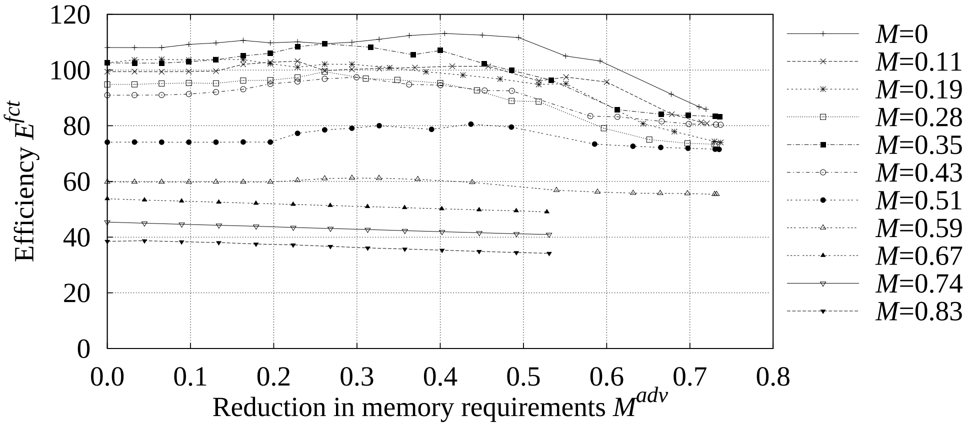


Figure 8.21: Influence of simultaneous subscription and advertisement pruning on the efficiency of the overlapping calculation function problem E^{fct}.

The reason for this influence of subscription pruning on the maximal reduction in the memory requirements for subscription routing tables M^{adv} is that the selection of advertisement pruning operations always considers the registered subscriptions. Because subscription pruning alters these registered subscriptions, advertisement pruning selects different pruning operations for different numbers of executed subscription pruning operations. For example, after executing all possible subscription pruning operations, advertisement pruning tends to leave over several predicates per advertisement (i.e., disjunctive nodes remain as root nodes because conjunctions could be pruned further). The underlying reason for this effect of subscription pruning on advertisement pruning is the structure of subscriptions and advertisements.

Efficiency

The behavior of the efficiency to solve the overlapping decision problem E^{dec} is illustrated in Figure 8.20. The development of the individual curves is on a par with our findings in Section 8.6.1: performing advertisement pruning improves the calculation efficiency. Considering the different stages of subscription pruning (represented by the different curves), the more subscription pruning operations are performed, the more efficient the calculation of overlap becomes. Without any advertisement pruning ($M^{adv} = 0$ on the abscissa), the overlapping decision problem is computed in $E^{dec} \approx 8.7$ milliseconds without subscription pruning (curve for $M = 0$). After all possible subscription pruning operations (curve for $M = 0.83$), the computation time improves to $E^{dec} \approx 1.8$ milliseconds.

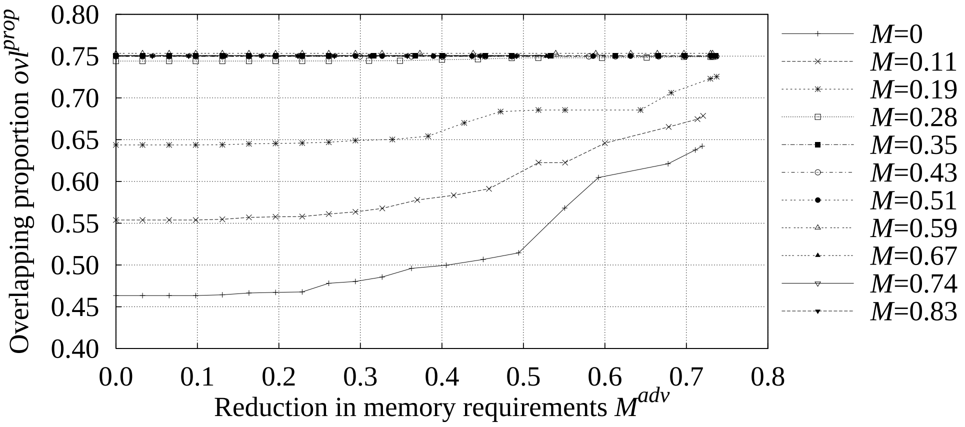


Figure 8.22: Influence of simultaneous subscription and advertisement pruning on the amount of overlap ovl^{prop}.

Analyzing the different curves individually, the advantageous effect of advertisement pruning on the calculation efficiency of the decision problem is highest if subscriptions are unpruned: With no subscription pruning, advertisement pruning improves the time to calculate the overlapping decision problem by 45 percent. After all subscription pruning (curve for $M = 0.83$), the improvement is reduced to 11 percent.

The reason for this behavior is the development of the overlapping relationships when pruning subscriptions: the more subscription pruning operations are executed, the higher the overlapping proportion ovl^{prop} becomes. The overlap does not increase further when additionally pruning advertisements. Hence the potential to avoid the evaluation of candidate subscriptions in the decision problem degrades with an increasing number of performed subscription pruning operations. The only effect of executing advertisement pruning then becomes the existence of fewer disjoint predicates.

Figure 8.21 shows the influence of subscription pruning and advertisement pruning on the efficiency of solving the overlapping function problem, E^{fct}. Generally, the more subscription pruning operations are performed, the more efficient the algorithm of solving the function problem. The average time per advertisement is situated between $E^{fct} \approx 108$ and $E^{fct} \approx 86$ milliseconds for no subscription pruning (curve for $M = 0$) and between $E^{fct} \approx 38$ and $E^{fct} \approx 34$ milliseconds after executing all subscription pruning operations (curve for $M = 0.83$). Hence the proportional efficiency improvement of the function problem calculation degrades when pruning subscriptions additionally to advertisements (as for the decision problem). The reason for this behav-

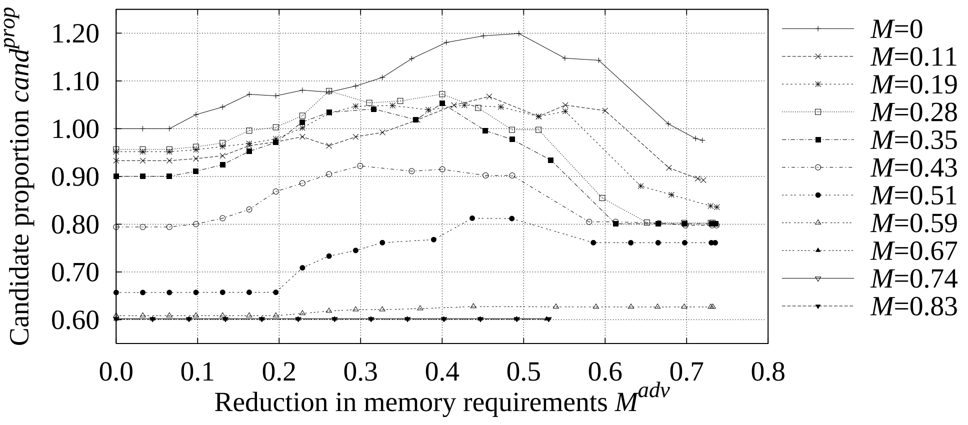


Figure 8.23: Influence of simultaneous subscription and advertisement pruning on the number of candidates proportional to the un-optimized setting $cand^{prop}$.

ior is the decrease in the number of disjoint predicates and the decrease in the number of candidate overlapping subscriptions when pruning subscriptions (cf. Figure 8.23).

Amount of Overlap

In this subsection, we explain the development of overlap between subscriptions and advertisements to emphasize the suitability of advertisement pruning for retaining the existing overlapping relationships within a pub-sub system. Subscription pruning, on the other hand, is expected to be unsuitable for this purpose due to its different target parameters.

Figure 8.22 shows the development of ovl^{prop} when pruning advertisements in its individual curves. Each curve represents a different stage of subscription pruning. Correlating these curves with each other thus shows the influence of subscription pruning on the overlap.

Figure 8.22 clearly reveals that advertisement pruning increases the overlapping proportion ovl^{prop} much less than subscription pruning: after subscription pruning reduces the event routing table size by approximately 28 percent ($M = 0.28$, fourth curve), the overlapping proportion increases to $ovl^{prop} \approx 0.75$ from its original (un-optimized) value of $ovl^{prop} \approx 0.46$. The overlapping proportion does not increase further when performing more subscription pruning (i.e., $ovl^{prop} \approx 0.75$ is the maximum for the registered subscriptions and advertisements).

Advertisement pruning, on the other hand, only marginally affects the overlap: when reducing the subscription routing table size by approximately 28 percent, the overlapping proportion only increases from $ovl^{prop} \approx 0.46$ to $ovl^{prop} \approx 0.48$ (compared to $ovl^{prop} \approx 0.75$ for subscription pruning).

Comparing the influences of subscription pruning and advertisement pruning the other way round, that is, considering the reduction in routing table size, a similar increase in overlap, for example, to $ovl^{prop} \approx 0.65$, is achieved by reducing the subscription routing table size by approximately 72 percent ($M^{adv} \approx 0.72$) when using advertisement pruning. However, for the same increase in overlap, the event routing table size is only reduced by approximately 19 percent ($M \approx 0.19$) when using subscription pruning.

Advertisement pruning therefore retains the existing overlapping relationships much more effectively than subscription pruning. This behavior fully meets our expectations and the design goals of advertisement pruning.

8.6.3 Findings for Advertisement Pruning

We evaluated the effects of advertisement pruning, the first designated advertisement-based routing optimization. We showed that the application of this optimization improves the efficiency of the calculation algorithm for both overlapping function and overlapping decision problem. Simultaneously, advertisement pruning decreases the sizes of subscription routing tables. These beneficial effects on the system are achieved with only slightly increasing the amount of overlap between subscriptions and advertisement.

This property of advertisement pruning directly corresponds to our design goal for this optimization, reflected in Part 2b of our central hypothesis. We proved this hypothesis in this section:

> *Advertisement pruning increases system efficiency and decreases routing table size, while only marginally affecting the existing overlap.*

In a second experiment we analyzed the system behavior if subscription pruning and advertisement pruning are applied at the same time. We found that also subscription pruning improves the efficiency properties of the overlapping calculation algorithms. However, subscription pruning strongly increases the existing overlapping relationships. This identified increase in overlap leads to a strongly growing number of forwarded subscriptions in the distributed system. These forwarded subscriptions, in turn, increase the size of event routing tables to a large extent.

We thus conclude that advertisement-based pub-sub systems should apply advertisement pruning rather than subscription pruning.

8.7 Summary

This chapter presented the results of an extensive empirical evaluation of our content-based pub-sub prototype BoP. The goals of the experiments were to analyze the behavior of our general Boolean approaches and to validate the claims of our central hypothesis.

We started by describing the experimental setup and methodology, and argued for the requirement of undertaking a real system analysis to obtain realistic test results. The experiments led to new insights for both pure subscription-based pub-sub systems, and subscription and advertisement-based systems: our event routing optimization, subscription pruning, results in an effective system optimization in combination with both general Boolean and restricted conjunctive subscriptions. All of the introduced pruning variants optimize the system with respect to their target parameters in comparison to the other variants.

Relating subscription pruning to subscription covering showed that subscription pruning fulfills our design goals and leads to an effective optimization regardless of the degree of cover. For common situations of cover among subscriptions, subscription pruning even realizes a distributed filtering process that is more space- and time-efficient than is achieved by subscription covering. Only in situations of extremely high cover, as expected, is subscription covering more suitable than subscription pruning. We believe that these situations of extreme covering proportions rarely occur in practice. Altogether, the results prove Part 2a of our central hypothesis.

Advertisement-based pub-sub systems solve the overlapping decision problem more efficiently in our Boolean computation approach than in a conjunctive computation approach. The Boolean approach is also more space-efficient for non-conjunctive subscriptions and advertisements. The calculation of the overlapping function problem shows a higher time efficiency in a conjunctive system. However, a conjunctive system requires more memory and thus does not scale to the same numbers of subscriptions and advertisements as the Boolean system.

Regarding optimizations for advertisement-based systems, we conclude that advertisement pruning effectively decreases the sizes of subscription routing tables. Advertisement pruning is the first designated interfering optimization for advertisement-based pub-sub systems. While performing its optimization, advertisement pruning only marginally increases the overlapping relationships in the system, as it is required for advertisement-based optimizations. With the help of our empirical experiments, we also prove Part 2b of our central hypothesis.

Chapter 9

Conclusion

IN THIS DISSERTATION, we have made the case for the general Boolean pub-sub model. In doing so, we developed and evaluated algorithms to support general Boolean subscriptions and advertisements in content-based pub-sub systems. Our focus was on general-purpose algorithms, that is, on system solutions that are applicable to a wide range of applications scenarios. The main hypothesis was split into two parts:

1. *In general-purpose pub-sub systems, a general Boolean filtering approach requires less memory and achieves higher filter efficiency than a conjunctive filtering approach.*

2. *The pruning of filter expressions is an effective routing optimization approach for both general Boolean subscriptions and advertisements.*

   (a) *Subscription pruning increases system efficiency and decreases routing table size, independently of the existing covering relationships.*

   (b) *Advertisement pruning increases system efficiency and decreases routing table size, while only marginally affecting the existing overlap.*

In the following section we summarize the steps we undertook for the proof of this hypothesis and outline our contributions to the research community. The summary is supplemented by Section 9.2 where we relate our findings to the broader pub-sub area and report observations from carrying out our research. Finally, we outline topics for future work that are raised by our research in Section 9.3.

9.1 Summary and Contributions

The overall structure of this dissertation reflects the different research areas that needed to be pursued to prove the hypothesis. Touching on various aspects of pub-sub systems, the different elements focus on state-of-the-art and related work (Chapter 2), application scenarios (Chapter 3), filtering in central broker components (Chapter 4 and Chapter 5), subscription-based event routing optimizations (Chapter 6), supporting advertisements (Chapter 7), advertisement-based routing optimizations (Chapter 7), and evaluating empirical studies (Chapter 8).

In detail, the chapters of this dissertation and their contributions can be summarized as follows:

Chapter 2: Background and Related Work. Chapter 2, as the foundation chapter, presents a study of related work in the pub-sub area. This chapter substantiates the rationale behind our hypothesis, and shows the common assumptions of current approaches and their implications.

We specifically decided to introduce content-based pub-sub systems from the viewpoint of database management systems to show both the similarity between these two kinds of systems and their differences. Based on this perspective, we argued that the conversion of Boolean to conjunctive subscriptions and advertisements that needs to be typically applied in pub-sub systems does not, in fact, have an equivalent counterpart in database management systems. Furthermore, the number of conversions in pub-sub systems is much higher than in database management systems due to the large number of registered subscriptions compared to the relatively small number of simultaneous queries. We thus concluded that the suitability of conversion in these systems is highly questionable, in particular for general-purpose system approaches.

Chapter 3: Application Scenario. The analysis of a real-world application scenario for content-based pub-sub mechanisms, online auctions, is developed in Chapter 3. Overall, the results of this chapter provide the community with a semi-realistic dataset for empirical experiments: for event messages, we analyzed the distributions of online book auctions on eBay. We also identified various subscription and advertisement classes that would typically be registered by auction site users. Our findings provide the basis for empirical studies in Chapter 8.

The constructed semi-realistic dataset for the evaluation of content-based pub-sub systems is the main contribution of Chapter 3. Using this dataset, evaluations of pub-sub systems reflect the system behavior in practical conditions. The results derived in experiments are realistic, in contrast to the synthetic findings of pure artificial settings which are used mostly today. The analysis reported in Chapter 3 constitutes an initial step towards pub-sub system evaluations in real-world settings, which is one of the major issues today.

Chapter 4: Filtering of General Boolean Subscriptions. Our proposal of a general-purpose event filtering algorithm for general Boolean subscriptions is the contribution of Chapter 4. In contrast to existing filtering approaches for this class of expressions, our algorithm applies predicate index structures that allow for an efficient filtering process. The algorithm extends the general-purpose conjunctive counting algorithm to a general Boolean filtering solution: candidate subscription matching, in an initial filtering step, identifies a set of potentially matching subscriptions; in the final filtering step, final subscription matching evaluates these candidates to identify matching subscriptions.

Chapter 4 also proposes various optimizations to the filtering algorithm. For example, the filtering shortcut optimization intertwines the matching algorithm and the routing algorithm; it eliminates the evaluation of most candidate subscriptions based on the current routing state of a filtering broker component.

Another important property of the filtering algorithm is the filtering of conjunctive subscriptions with only marginal overhead compared to the original conjunctive counting approach. Altogether, our algorithm allows for the direct internal support of general Boolean subscriptions in the central filtering components of content-based pub-sub systems.

Chapter 5: Comparison of Boolean and Conjunctive Filtering. Chapter 5 provides a comparative evaluation of our Boolean filtering algorithm (see Chapter 4) and a traditional general-purpose conjunctive approach. In the theoretical part of the chapter, we introduce a characterization framework for subscriptions, both general Boolean and conjunctive subscriptions. This framework permits the description of the memory requirements of filtering algorithms based on 13 parameters.

We described the memory use of the counting algorithm, our Boolean algorithm, and the cluster algorithm. Subsequently, we compared the memory requirements of these algorithms. We showed that the occurrence of only one disjunction in subscriptions already favors our Boolean filtering solution. We thus proved Part 1 of our central hypothesis with respect to memory requirements. Our characterization framework permits the determination of the filtering algorithm with the smallest memory requirements for any given subscriptions.

The practical part of Chapter 5 verified our theoretical findings by analyzing the memory requirements of the counting algorithm and the Boolean filtering algorithm under a real implementation. We compared these requirements for a large number of system settings to validate the predictions of our characterization framework. Finally, we compared the filter efficiency of the counting and Boolean algorithms for typical subscriptions of the auctioning scenario. We showed that the Boolean algorithm filters these classes more efficiently than its conjunctive counterpart. We therefore also proved Part 1 of our central hypothesis with respect to filter efficiency.

Chapter 6: Routing Optimizations for General Boolean Subscriptions. The direct support of general Boolean subscriptions in distributed pub-sub systems is the content of Chapter 6. We focus on event routing optimizations—current approaches are not applicable to Boolean subscriptions—and present two optimizations: predicate replacement and subscription pruning. Both approaches follow optimization objectives that are orthogonal to recent approaches. Whereas current optimizations depend on similarities and covering relationships among subscriptions, our proposals actively alter individual routing entries regardless of such relationships.

We identified subscription pruning as having a high optimization potential. Moreover subscription pruning offers the opportunity to optimize pub-sub systems with respect to different target parameters. We consequently tailored this optimization with respect to three target parameters: network load, filter efficiency, and memory use. For the network load, we developed two measures, one based on subscription accuracy alone, and another based on subscription accuracy and predicate occurrence. We additionally presented the theoretical foundations for two extended measures.

In concluding Chapter 6 we described how to practically implement subscription pruning within pub-sub systems. We identified three variants of its realization: post-pruning, pre-pruning, and combined pruning.

Chapter 7: Supporting General Boolean Advertisements. The final milestone to provide for the general Boolean pub-sub model is presented in Chapter 7: the support of general Boolean advertisements. The chapter is split into two parts. The first part focuses on the calculation of the overlapping relationships between general Boolean subscriptions and advertisements; the second part introduces the first designated advertisement-based routing optimization for content-based pub-sub systems, advertisement pruning.

Our approach to calculating the overlapping relationships extends the conjunctive computation algorithm for the overlapping task. Similar to the Boolean filtering approach (see Chapter 4), our algorithm firstly determines a set of overlapping candidates, that is, subscriptions or advertisements that overlap potentially. Secondly, these candidates are analyzed to detect whether they constitute an overlap. The algorithm handles restricted conjunctive expressions in the same way as a specialized conjunctive solution.

Advertisement pruning is tailored to optimize advertisements, that is, entries in subscription routing tables. When optimizing, it attempts to alter the existing overlapping relationships in the system as little as possible. This avoids a blow-up of subscription routing tables. Advertisement pruning is the first optimization approach that specifically targets the optimization of subscription routing tables and thus constitutes pioneering work in this area.

Chapter 8: Experimental Evaluation. Chapter 8 includes the results of our extensive practical analysis of our distributed pub-sub prototype, of our routing optimizations (Chapter 6 and Chapter 7) for general Boolean subscriptions and advertisements, and of our overlapping calculation algorithm (Chapter 7). We compare a distributed and optimized general Boolean system against a distributed and optimized conjunctive system, extending our work from Chapter 5.

We evaluated a true system setting for the dataset derived in Chapter 3. Analyzing a real system configuration allowed us to consider both system-specific parameters (e.g., incorporating the influence of the processor cache and the existing network) but also general parameters (e.g., sizes of routing tables and network load) in our experiments. This real system analysis was extended by a simulation to investigate the behavior for large network scales.

The main findings of our experiments are as follows:

1. The four introduced measures to select subscription pruning operations

fulfill their individual design goals of optimizing with respect to memory requirements, filter efficiency, and network load.

2. Subscription pruning optimizes the system with respect to space and time efficiency regardless of the degree of cover among subscriptions (proving Part 2a of our central hypothesis). It even optimizes the system when the full potential of subscription covering has been reached.
3. Subscription pruning in combination with the Boolean filtering approach leads to a more time-efficient and space-efficient filtering and routing process than subscription covering in combination with the conjunctive counting approach for non-extreme covering proportions.
4. The general Boolean algorithm to calculate the overlapping decision problem is more time-efficient than the conjunctive calculation algorithm.
5. Advertisement pruning only marginally increases the overlapping relationships between subscriptions and advertisements (compared to subscription pruning), while reducing memory requirements and increasing system efficiency (proving Part 2b of our central hypothesis).

Summary of Contributions. In summary, the main contributions of this dissertation are:

- The provision of a *dataset* for experimental evaluations of content-based pub-sub systems that is derived from the typical characteristics of online auctions.
- The development of the first *filtering algorithm* for general Boolean subscriptions that applies one-dimensional predicate indexes for efficiency reasons.
- The introduction of a *classification framework* for subscriptions that allows for the decision whether to apply a general-purpose conjunctive or a general-purpose general Boolean filtering algorithm with respect to their memory requirements.
- The proposal of the first *event routing optimization*, subscription pruning, that is practically applicable to general Boolean subscriptions but can also be used in addition to recent optimization approaches.

- The development of an *algorithm* to calculate the *overlap* between general Boolean subscriptions and advertisements.

- The introduction of the first *advertisement-based routing optimization*, advertisement pruning, that is tailored to optimize (general Boolean) advertisements.

- The derivation from our classification framework that the occurrence of *only one disjunction* in subscriptions already favors a Boolean filtering approach with respect to memory usage.

- The provision of *empirical evidence* that

  - the predictions of the *classification framework* with respect to memory requirements *hold in practice*.
  - the proposed *filtering algorithm* (without routing optimizations) filters more efficiently than its conjunctive counterpart.
  - our *event routing optimization*, subscription pruning, optimizes the system regardless of the existing degree of cover.
  - our Boolean *filtering approach in combination with subscription pruning* leads to a more efficient filtering process than the conjunctive counting approach in combination with subscription covering (except in case of high covering proportions).
  - our *overlapping calculation* approach solves the overlapping decision problem more efficiently than a conjunctive solution.
  - our *advertisement-based routing optimization*, advertisement pruning, maintains the existing overlapping relationships in contrast to other optimizations.

9.2 Observations

In two previous research projects [BH04, HB02], we noticed the focus of most of the work in the pub-sub area on conjunctive expressions. We also observed that realistic, widely used high-level applications for content-based pub-sub systems were still missing. We believe that this situation led to the artificial experimental settings often used today.

The issue of missing applications and practical implementations was also identified and criticized by Rosenblum at DOA[1] 2005 [Ros05], and by the pub-sub community at DEBS[2] 2006 [DEB06] and Event Processing [Dag07]. We believe that this lack of real-world applications has contributed to the focus of pub-sub research on conjunctive expressions. For example, a popular setting in the literature is the stock broker application, which simplifies the content-based pub-sub paradigm to the selection of stocks of certain companies. Potentially, subscriptions further restrict the prices of the stocks of interest, leading to conjunctive subscriptions of two predicates.

However, an analysis of more sophisticated settings reveals the need to support more complex subscriptions. In this dissertation, we made a first step for the consideration of such scenarios with our choice of an online auction example application. The chosen application, evidently, influences the test settings that are applied in conducted studies. We believe that our analysis of typical event distributions and the identification of various subscription and advertisement classes is a step in the direction of more realistic experiments and evaluations. This focus on existing advanced problems should lead to a wider adoption of the pub-sub paradigm in everyday systems.

In the dissertation we demonstrated that the reconsideration of initial assumptions can result in important novel findings. Our algorithms for filtering, routing, and overlapping calculation are applicable to all kinds of Boolean subscriptions. Thus this dissertation contributes valuable findings for both conjunctive and general Boolean application scenarios.

One may argue that the application of low-level optimizations in conjunctive approaches might change the outcome of our experimental evaluation. However, these optimizations are similarly applicable to our Boolean solutions; such low-level considerations are not the focus of this work. In this dissertation, we specifically aimed at deriving general conclusions about the applicability and behavior of general-purpose general Boolean and conjunctive solutions for high-level application scenarios.

One might take the position that existing conjunctive approaches do not target the general Boolean case, although some authors particularly state that the canonical conversion allows for the focus on conjunctions, as shown in Section 1.2 (page 4). We do not question that some work on conjunctive expressions explicitly and solely targets this restricted setting. The research

[1]International Symposium on Distributed Objects and Applications.

[2]The leading international forum for research into event-based computing.

in this dissertation, however, focuses on the general Boolean scenario and thus complements this work on conjunctions.

It is to be seen whether the issue of the validity of assumptions for both applications and experiments will be resolved once content-based pub-sub systems are applied in practice in large scale. However, if this implementation is undertaken by commercial companies, the research community might still lack the required realistic real-world datasets for different applications. It would thus be preferable if the whole community stands together to build and maintain a large-scale content-based pub-sub system. If this system is adopted for real-world applications, valuable datasets and insights into applications would become publicly available.

9.3 Future Work

Content-based pub-sub systems is an active research area. Various projects focus on the extension of the core pub-sub functionality (see Section 2.1.3, page 22). Although these extensions might ultimately enhance the application fields for pub-sub by providing extended employment mechanisms, we do not aim at naming such extensions in this section. Instead we focus on those topics that are directly raised by the research presented in this dissertation.

Combination of Routing Optimizations. In Section 8.4.2 (page 269), we showed that subscription pruning can be applied additionally to subscription covering. This advantageous property results from the orthogonal optimization principles of these two approaches (see Section 6.1, page 150). The presented, step-wise application of firstly covering and secondly pruning (see Section 8.4.2) is only one option to combine routing optimizations. We believe that there is further potential if utilizing multiple optimizations simultaneously: the application of subscription pruning might lead to more overlap among subscriptions and, potentially, to more covering relationships among them. Hence, after having applied subscription pruning, covering might become a more appropriate optimization than before performing pruning. In particular, there is further room to design extended pruning measures that aim at increasing the covering relationships.

Furthermore, pruning can be applied to solve the imperfect merging task (see Section 6.3.5, page 161). In this case, either the presented (see Section 6.4, page 162) or extended pruning measures can be applied to merge several subscriptions. Due to the generality of subscription pruning there is a large potential to tailor the optimization to this application of merging.

In Section 6.2 (page 152), we also introduced predicate replacement as a novel optimization approach. We mainly focused on pruning due to its high optimization potential. Future work could try to combine both proposals, as indicated in Section 6.3.4 (page 160). With respect to covering, predicate replacement can naturally be applied to effectively create covering subscriptions.

Implementation of Pruning. Within this dissertation our focus was the evaluation of the optimization potential of subscription pruning. The practical application of this optimization requires the ordering of pruning options. This ordering can be efficiently realized by a priority queue, in particular if performing bulk pruning (see Section 6.6.2, page 193). Our prototype BoP currently recalculates the ranking value for each subscription after it is pruned.

Future work could try to optimize this recalculation process and only recompute the partial ranking of those parts of subscriptions or advertisements that are affected by pruning. Further optimization potential might exist if specializing the applied priority queue, for example, in the accuracy-based pruning variant the ranking measure never decreases due to pruning.

Another branch of future research that is raised by our work regards the dynamic determination of the cut-off point for subscription and advertisement pruning. This dissertation investigated the general potential of subscription and advertisement pruning as routing optimizations. In the current version of BoP, the cut-off point for pruning operations needs to be set statically by a system administrator. Even though accuracy and commonality-based pruning (see Section 6.4.5, page 180) leads to a minor cut-off point only, a dynamic determination of this point remains a worthwhile question for future research.

Characterization Framework for Filter Efficiency. We introduced a characterization framework for subscriptions. This framework considers those properties of subscriptions that influence the memory requirements of general-purpose filtering algorithms in Chapter 5 (page 119). Based on the proposed framework one can determine whether a conjunctive or a Boolean filtering solution should be applied for a given set of subscriptions.

An exciting topic for future work is to develop an orthogonal characterization framework to describe those attributes of subscriptions that influence the filter efficiency properties of different algorithms. Having identified these attributes and described various algorithms based on the framework, a comparison of these descriptions allows for the determination of the preferable algorithm for a given subscription set (similar to our method in Section 5.5, page 132). The efficiency-based framework might then have to consider attributes of event messages, which apparently influence filter efficiency as well.

Setting the potential of this framework into perspective allows the creation of a pub-sub system that automatically adapts to the current subscription and event load. The system could alter the applied filtering algorithm based on its current configuration. That is, it could aim at the most space-efficient, the most time-efficient, or the optimally balanced filtering solution.

Test Suites and Application Specifics. Taking up the issue about the validity of assumptions and evaluations from our observations (Section 9.2, page 295), the pub-sub community requires realistic test suites for different application areas. These test suites need to contain the specifics of typical event loads, subscriptions, and advertisements. In this dissertation, we undertook a first step in this direction, using the online auction scenario. The information retrieval community with its Cross-Language Evaluation Forum (CLEF[3]) and Text REtrieval Conference (TREC[4]) could serve as a role model for such test infrastructures for the pub-sub area.

It is evident that the whole community and industry should stand together in this effort to creating realistic test settings for different applications. Otherwise, it remains questionable whether the developed test suites will become widely accepted. Moreover, the effort of developing test settings should stretch across all sub-communities of pub-sub. Taking such a comprehensive approach allows for the incorporation of high-level requirements—for example, user interface and user experience aspects—into low-level requirements, for example, internal subscription definition language and data model. Furthermore, it might reveal the independence of the overall pub-sub system from realization specifics, for example, whether to apply an attribute-value pair or an XML event model.

[3]`http://www.clef-campaign.org/`
[4]`http://trec.nist.gov/`

Appendix A

Distributions of Event Messages

In this appendix, we give details about the distributions of attribute values in event messages. The findings we present in the following sections are derived from our analysis of eBay (see Chapter 3). In Appendix A.1, we name the domains of those attributes that are defined as enumerations. Appendix A.2 specifies the distributions of the possible values of the eight attributes of event messages in the online auction application scenario (see Section 3.2, page 72).

A.1 Attribute Domains for Enumerations

In this section, we state the domains of the attributes `Category`, `Format`, `Special Attribute`, and `Condition`. These four attributes specify enumerations as their domains.

The twenty-two possible values for the categories of fiction books (i.e., attribute `Category`) are as follows:

1. Action, Adventure
2. Classics
3. Fantasy
4. Folklore, Mythology
5. Historical
6. Horror
7. Humor

8. Literary Collections
9. Literary Criticism
10. Literature, Ancient
11. Literature, Classic
12. Literature, Modern
13. Military
14. Mystery, Thriller
15. Plays, Screenplays
16. Poetry
17. Pulps
18. Religious, Inspirational
19. Romance
20. Science Fiction
21. Westerns
22. Other

Sellers can specify four different formats (i.e., attribute `Format`) for books:

1. Hardcover
2. Softcover
3. Mixed Lot
4. Other

Special attributes (i.e., attribute `Special Attribute`) might increase the value of books. Sellers can choose among the following options:

1. 1st Edition
2. Signed

3. Unspecified

The condition (i.e., attribute `Condition`) of a book is one of the following:

1. New
2. Used

A.2 Distributions of Attribute Values

We give an overview of the exact probabilities of all possible combinations of the values of the attributes `Category`, `Format`, `Special Attribute`, and `Condition` in Tables A.1 to A.6. Each row of these tables specifies a category; the four right columns state a combination of the values of the attributes `Condition`, `Special Attribute`, and `Format`: Table A.1 contains the probabilities for used books of the first edition, Table A.2 for signed used books, Table A.3 for all other used books, Table A.4 for new books of the first edition, Table A.5 for signed new books, and Table A.6 for all other new books.

For attribute `Buy It Now`, we analyzed the probability that an item of any category is sold as Buy-It-Now item. In all other cases, the book is not a Buy-It-Now item. We give an overview of the results in Table A.7.

We also analyzed the number of bids depending on the category of items. An overview of our results is given in Tables A.8 to A.11: Table A.8 contains the results for 0 to 3 bids, Table A.9 for 4 to 7 bids, Table A.10 for 8 to 20 bids, and Table A.11 for 21 or more bids.

The probability of auction items of all 22 categories depending on attribute `Price` is shown in Tables A.12 to A.15: Table A.12 gives an overview for prices up to \$4.00, Table A.13 for prices ranging from \$4.01 to \$8.00, Table A.14 for prices in between \$8.01 and \$30.00, and Table A.15 covers prices over \$30.01.

Table A.16 gives an overview of the total number of items for all categories.

Table A.1: Overview of the probabilities of used books of the first edition for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	6.469393×10^{-6}	6.469393×10^{-6}	3.234697×10^{-5}	3.752248×10^{-4}
2	0.000000	0.000000	0.000000	4.528575×10^{-5}
3	0.000000	6.469393×10^{-6}	1.293879×10^{-4}	2.911227×10^{-4}
4	0.000000	0.000000	1.293879×10^{-5}	6.469393×10^{-6}
5	6.469393×10^{-6}	0.000000	7.763272×10^{-5}	1.487960×10^{-4}
6	1.293879×10^{-5}	0.000000	5.175515×10^{-5}	3.816942×10^{-4}
7	1.940818×10^{-5}	0.000000	1.552654×10^{-4}	1.940818×10^{-4}
8	6.469393×10^{-6}	0.000000	0.000000	6.469393×10^{-5}
9	0.000000	0.000000	0.000000	6.469393×10^{-6}
10	0.000000	0.000000	0.000000	0.000000
11	0.000000	0.000000	0.000000	1.940818×10^{-5}
12	2.587757×10^{-5}	0.000000	1.682042×10^{-4}	7.633884×10^{-4}
13	0.000000	0.000000	1.293879×10^{-5}	1.811430×10^{-4}
14	4.528575×10^{-5}	0.000000	1.423267×10^{-4}	1.390920×10^{-3}
15	0.000000	0.000000	3.234697×10^{-5}	6.469393×10^{-6}
16	2.587757×10^{-5}	6.469393×10^{-6}	1.293879×10^{-4}	2.587757×10^{-4}
17	0.000000	0.000000	6.469393×10^{-6}	6.469393×10^{-6}
18	0.000000	0.000000	6.469393×10^{-5}	9.057151×10^{-5}
19	0.000000	0.000000	2.070206×10^{-4}	1.811430×10^{-4}
20	1.293879×10^{-5}	0.000000	2.846533×10^{-4}	3.105309×10^{-4}
21	6.469393×10^{-6}	0.000000	3.881636×10^{-5}	6.469393×10^{-5}
22	1.035103×10^{-4}	6.469393×10^{-6}	1.940818×10^{-4}	6.210618×10^{-4}

Table A.2: Overview of the probabilities of signed used books for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	1.811430×10^{-4}	1.293879×10^{-5}	1.759675×10^{-3}	8.584885×10^{-3}
2	2.587757×10^{-5}	6.469393×10^{-6}	2.781839×10^{-4}	1.261532×10^{-3}
3	7.116333×10^{-5}	4.528575×10^{-5}	3.836350×10^{-3}	5.304903×10^{-3}
4	1.293879×10^{-5}	6.469393×10^{-6}	1.552654×10^{-4}	6.340005×10^{-4}
5	1.293879×10^{-4}	0.000000	5.757760×10^{-4}	3.111778×10^{-3}
6	1.164491×10^{-4}	2.587757×10^{-5}	1.953757×10^{-3}	5.272556×10^{-3}
7	9.057151×10^{-5}	0.000000	1.468552×10^{-3}	2.613635×10^{-3}
8	1.293879×10^{-5}	0.000000	1.487960×10^{-4}	5.951842×10^{-4}
9	1.293879×10^{-5}	0.000000	4.528575×10^{-5}	1.552654×10^{-4}
10	0.000000	0.000000	6.469393×10^{-6}	4.528575×10^{-5}
11	2.587757×10^{-5}	0.000000	1.293879×10^{-5}	4.140412×10^{-4}
12	4.075718×10^{-4}	6.469393×10^{-6}	1.222715×10^{-3}	6.922251×10^{-3}
13	1.293879×10^{-5}	1.293879×10^{-5}	8.022048×10^{-4}	1.766144×10^{-3}
14	2.717145×10^{-4}	9.057151×10^{-5}	2.464839×10^{-3}	1.676220×10^{-2}
15	3.234697×10^{-5}	0.000000	2.587757×10^{-4}	4.011024×10^{-4}
16	7.763272×10^{-5}	6.469393×10^{-6}	5.563678×10^{-4}	1.255062×10^{-3}
17	6.469393×10^{-6}	0.000000	3.364085×10^{-4}	7.116333×10^{-5}
18	4.528575×10^{-5}	1.293879×10^{-5}	2.652451×10^{-4}	8.798375×10^{-4}
19	1.293879×10^{-4}	5.822454×10^{-5}	6.165332×10^{-3}	4.535045×10^{-3}
20	1.811430×10^{-4}	3.234697×10^{-5}	6.197679×10^{-3}	5.686597×10^{-3}
21	4.528575×10^{-5}	1.293879×10^{-5}	1.300348×10^{-3}	1.352103×10^{-3}
22	7.181027×10^{-4}	4.528575×10^{-5}	3.791064×10^{-3}	7.925007×10^{-3}

Table A.3: Overview of the probabilities of used books not specified as signed and first edition for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	7.569190×10^{-4}	3.881636×10^{-4}	2.761784×10^{-2}	2.664743×10^{-2}
2	2.587757×10^{-4}	9.704090×10^{-5}	6.275312×10^{-3}	7.879721×10^{-3}
3	3.881636×10^{-4}	4.593269×10^{-4}	2.851709×10^{-2}	1.143789×10^{-2}
4	9.704090×10^{-5}	1.293879×10^{-5}	1.889063×10^{-3}	2.160777×10^{-3}
5	3.493472×10^{-4}	9.704090×10^{-5}	8.759559×10^{-3}	8.112619×10^{-3}
6	4.140412×10^{-4}	4.269800×10^{-4}	2.172422×10^{-2}	1.339811×10^{-2}
7	3.687554×10^{-4}	1.617348×10^{-4}	1.653577×10^{-2}	7.892660×10^{-3}
8	9.704090×10^{-5}	2.587757×10^{-5}	1.345634×10^{-3}	2.775370×10^{-3}
9	2.587757×10^{-5}	6.469393×10^{-6}	2.975921×10^{-4}	4.011024×10^{-4}
10	3.881636×10^{-5}	0.000000	1.099797×10^{-4}	2.070206×10^{-4}
11	1.293879×10^{-4}	5.175515×10^{-5}	1.811430×10^{-3}	2.801247×10^{-3}
12	7.827966×10^{-4}	2.781839×10^{-4}	1.564299×10^{-2}	1.404505×10^{-2}
13	8.410211×10^{-5}	1.164491×10^{-4}	6.294720×10^{-3}	4.011024×10^{-3}
14	1.552654×10^{-3}	1.973165×10^{-3}	5.815985×10^{-2}	4.951674×10^{-2}
15	4.528575×10^{-5}	3.881636×10^{-5}	1.973165×10^{-3}	1.119205×10^{-3}
16	2.070206×10^{-4}	3.234697×10^{-5}	2.607166×10^{-3}	4.030432×10^{-3}
17	6.469393×10^{-6}	0.000000	1.377981×10^{-3}	2.393676×10^{-4}
18	3.752248×10^{-4}	5.175515×10^{-4}	1.106266×10^{-2}	6.236495×10^{-3}
19	8.151436×10^{-4}	1.462083×10^{-3}	1.139907×10^{-1}	2.366504×10^{-2}
20	6.534087×10^{-4}	6.275312×10^{-4}	3.753542×10^{-2}	1.460142×10^{-2}
21	1.552654×10^{-4}	8.410211×10^{-5}	7.743864×10^{-3}	5.136698×10^{-3}
22	1.669103×10^{-3}	6.728169×10^{-4}	5.508041×10^{-2}	2.617517×10^{-2}

Table A.4: Overview of the probabilities of new books of the first edition for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	1.293879×10^{-5}	0.000000	1.682042×10^{-4}	5.240209×10^{-4}
2	0.000000	6.469393×10^{-6}	2.587757×10^{-5}	1.423267×10^{-4}
3	5.175515×10^{-4}	1.293879×10^{-5}	4.075718×10^{-4}	1.125674×10^{-3}
4	0.000000	0.000000	3.234697×10^{-5}	5.175515×10^{-5}
5	3.234697×10^{-5}	0.000000	6.469393×10^{-5}	1.358573×10^{-4}
6	4.528575×10^{-5}	0.000000	2.264288×10^{-4}	7.504496×10^{-4}
7	6.469393×10^{-6}	0.000000	1.035103×10^{-4}	1.552654×10^{-4}
8	0.000000	0.000000	1.293879×10^{-5}	9.057151×10^{-5}
9	0.000000	0.000000	0.000000	0.000000
10	0.000000	0.000000	6.469393×10^{-6}	0.000000
11	0.000000	0.000000	0.000000	6.469393×10^{-6}
12	7.116333×10^{-5}	1.293879×10^{-5}	3.622860×10^{-4}	1.268001×10^{-3}
13	0.000000	0.000000	3.881636×10^{-5}	1.164491×10^{-4}
14	1.035103×10^{-4}	0.000000	1.746736×10^{-4}	9.898172×10^{-4}
15	6.469393×10^{-6}	0.000000	1.293879×10^{-5}	0.000000
16	2.587757×10^{-5}	0.000000	9.704090×10^{-5}	5.175515×10^{-5}
17	0.000000	0.000000	1.293879×10^{-5}	0.000000
18	1.293879×10^{-5}	0.000000	1.940818×10^{-5}	4.528575×10^{-5}
19	2.587757×10^{-5}	0.000000	1.229185×10^{-4}	7.763272×10^{-5}
20	3.234697×10^{-4}	0.000000	2.264288×10^{-4}	6.081230×10^{-4}
21	0.000000	0.000000	0.000000	3.881636×10^{-5}
22	2.070206×10^{-4}	0.000000	2.587757×10^{-4}	9.121845×10^{-4}

Table A.5: Overview of the probabilities of signed new books for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	4.528575×10^{-5}	0.000000	6.857557×10^{-4}	3.098839×10^{-3}
2	0.000000	1.293879×10^{-5}	7.116333×10^{-5}	2.587757×10^{-4}
3	6.275312×10^{-4}	1.293879×10^{-5}	1.863185×10^{-3}	3.706962×10^{-3}
4	6.469393×10^{-6}	6.469393×10^{-6}	1.035103×10^{-4}	2.717145×10^{-4}
5	4.528575×10^{-5}	0.000000	3.493472×10^{-4}	7.763272×10^{-4}
6	1.035103×10^{-4}	6.469393×10^{-6}	7.504496×10^{-4}	1.999043×10^{-3}
7	2.587757×10^{-5}	0.000000	4.722657×10^{-4}	9.510008×10^{-4}
8	1.940818×10^{-5}	0.000000	4.528575×10^{-5}	3.493472×10^{-4}
9	0.000000	0.000000	6.469393×10^{-6}	2.587757×10^{-5}
10	0.000000	0.000000	6.469393×10^{-6}	6.469393×10^{-6}
11	0.000000	0.000000	1.293879×10^{-5}	1.940818×10^{-5}
12	1.487960×10^{-4}	1.293879×10^{-5}	1.442675×10^{-3}	2.969452×10^{-3}
13	1.293879×10^{-5}	0.000000	1.099797×10^{-4}	3.816942×10^{-4}
14	1.552654×10^{-4}	1.293879×10^{-5}	9.057151×10^{-4}	4.839106×10^{-3}
15	0.000000	0.000000	7.116333×10^{-5}	1.293879×10^{-5}
16	3.881636×10^{-5}	0.000000	2.652451×10^{-4}	2.264288×10^{-4}
17	0.000000	0.000000	3.234697×10^{-5}	6.469393×10^{-6}
18	6.469393×10^{-6}	0.000000	2.199594×10^{-4}	4.593269×10^{-4}
19	4.528575×10^{-5}	0.000000	1.274470×10^{-3}	1.196838×10^{-3}
20	3.558166×10^{-4}	0.000000	2.561880×10^{-3}	2.814186×10^{-3}
21	0.000000	0.000000	2.005512×10^{-4}	2.328982×10^{-4}
22	2.911227×10^{-4}	0.000000	1.953757×10^{-3}	2.600696×10^{-3}

Table A.6: Overview of the probabilities of new books neither specified as signed nor specified as first edition for all categories (given in the rows) and formats (given in the columns). The first column contains the category, abbreviated by "C".

C	Other format	Mixed lot	Softcover	Hardcover
1	3.364085×10^{-4}	0.000000	4.366840×10^{-3}	9.568233×10^{-3}
2	5.693066×10^{-4}	1.293879×10^{-5}	9.315926×10^{-4}	1.022164×10^{-3}
3	9.121845×10^{-4}	7.116333×10^{-5}	5.654250×10^{-3}	6.779924×10^{-3}
4	7.116333×10^{-5}	1.940818×10^{-5}	5.822454×10^{-4}	8.022048×10^{-4}
5	1.940818×10^{-4}	6.469393×10^{-6}	1.694981×10^{-3}	1.779083×10^{-3}
6	2.328982×10^{-4}	3.234697×10^{-5}	3.137656×10^{-3}	3.855758×10^{-3}
7	2.199594×10^{-4}	6.469393×10^{-6}	3.001798×10^{-3}	3.157064×10^{-3}
8	9.057151×10^{-5}	0.000000	3.428778×10^{-4}	7.310414×10^{-4}
9	0.000000	0.000000	1.423267×10^{-4}	6.986945×10^{-4}
10	0.000000	0.000000	1.293879×10^{-4}	1.229185×10^{-4}
11	1.293879×10^{-5}	0.000000	4.075718×10^{-4}	3.558166×10^{-4}
12	2.587757×10^{-4}	2.587757×10^{-5}	4.218044×10^{-3}	4.593269×10^{-3}
13	5.175515×10^{-5}	0.000000	6.340005×10^{-4}	8.086742×10^{-4}
14	5.757760×10^{-4}	6.469393×10^{-5}	7.491557×10^{-3}	1.683336×10^{-2}
15	4.528575×10^{-5}	0.000000	4.011024×10^{-4}	8.410211×10^{-5}
16	1.099797×10^{-4}	0.000000	7.827966×10^{-4}	8.086742×10^{-4}
17	2.587757×10^{-5}	0.000000	9.057151×10^{-5}	6.469393×10^{-6}
18	1.617348×10^{-4}	6.469393×10^{-6}	2.988860×10^{-3}	2.937105×10^{-3}
19	1.940818×10^{-4}	2.587757×10^{-5}	1.126968×10^{-2}	6.715230×10^{-3}
20	9.380620×10^{-4}	3.234697×10^{-5}	7.038700×10^{-3}	6.042413×10^{-3}
21	5.822454×10^{-5}	0.000000	3.946330×10^{-4}	5.563678×10^{-4}
22	8.086742×10^{-4}	2.523063×10^{-4}	1.195544×10^{-2}	8.947171×10^{-3}

Table A.7: Overview of the probabilities of books specified as Buy-It-Now items (`Buy It Now` = `Yes`) for all categories (given in the rows).

Category	Buy It Now
1	0.148564
2	0.137605
3	0.143228
4	0.162313
5	0.116194
6	0.177406
7	0.120268
8	0.136015
9	0.173759
10	0.314286
11	0.186170
12	0.120437
13	0.127722
14	0.169996
15	0.133903
16	0.101506
17	0.043605
18	0.164380
19	0.182857
20	0.135013
21	0.113257
22	0.138597

Table A.8: Overview of the probabilities of book items for all categories (given in the rows) having 0 to 3 bids (given in the columns).

Category	0 bids	1 bid	2 bids	3 bids
1	0.890891	0.071063	0.014498	0.006369
2	0.856820	0.094574	0.019970	0.007913
3	0.814717	0.119045	0.024911	0.011824
4	0.840345	0.117584	0.014024	0.011866
5	0.873781	0.092817	0.014189	0.007685
6	0.861950	0.089958	0.017621	0.009009
7	0.847547	0.100000	0.020377	0.007170
8	0.871152	0.093501	0.018244	0.004561
9	0.891566	0.096386	0.008032	0.004016
10	0.835294	0.129412	0.023529	0.011765
11	0.866071	0.088170	0.014509	0.008929
12	0.866002	0.091617	0.016828	0.007323
13	0.868811	0.081164	0.012762	0.010720
14	0.884921	0.082129	0.012797	0.006242
15	0.887755	0.085034	0.017007	0.006803
16	0.880086	0.092077	0.009279	0.009279
17	0.815884	0.133574	0.021661	0.007220
18	0.823123	0.106225	0.023715	0.014575
19	0.859675	0.096295	0.017726	0.008275
20	0.850492	0.103581	0.018532	0.008505
21	0.828596	0.115872	0.024486	0.009182
22	0.869191	0.090694	0.016185	0.008324

Table A.9: Overview of the probabilities of book items for all categories (given in the rows) having 4 to 7 bids (given in the columns).

Category	4 bids	5 bids	6 bids	7 bids
1	0.005950	0.003771	0.001760	0.001844
2	0.004145	0.006405	0.003391	0.001507
3	0.008151	0.005510	0.004018	0.003444
4	0.004315	0.004315	0.002157	0.003236
5	0.003547	0.001774	0.002069	0.001478
6	0.005962	0.004240	0.002782	0.002517
7	0.006226	0.006604	0.005472	0.002075
8	0.003421	0.003421	0.000000	0.003421
9	0.000000	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000	0.000000
11	0.006696	0.002232	0.005580	0.001116
12	0.005142	0.004363	0.004207	0.000779
13	0.008167	0.005615	0.003573	0.002552
14	0.003924	0.003567	0.002051	0.001204
15	0.001701	0.000000	0.000000	0.000000
16	0.001428	0.002855	0.001428	0.000714
17	0.003610	0.003610	0.000000	0.003610
18	0.008646	0.006423	0.005435	0.004447
19	0.006263	0.004289	0.001860	0.001632
20	0.005103	0.004834	0.003133	0.002596
21	0.006996	0.004373	0.004373	0.001749
21	0.005318	0.003295	0.002081	0.001561

Table A.10: Overview of the probabilities of book items for all categories (given in the rows) having 8 to 20 bids (given in the columns).

Category	8 bids	9 bids	10 bids	11 to 20 bids
1	0.001006	0.000251	0.000587	0.001676
2	0.001130	0.000377	0.000754	0.002261
3	0.001492	0.000918	0.001837	0.004018
4	0.000000	0.001079	0.000000	0.001079
5	0.000000	0.000591	0.000591	0.000887
6	0.001722	0.001325	0.001325	0.001590
7	0.001698	0.000755	0.000943	0.001132
8	0.000000	0.001140	0.001140	0.000000
9	0.000000	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000	0.000000
11	0.000000	0.002232	0.001116	0.002232
12	0.000779	0.000779	0.001246	0.000779
13	0.001531	0.001021	0.001531	0.002042
14	0.000803	0.000713	0.000535	0.000936
15	0.000000	0.000000	0.000000	0.001701
16	0.000000	0.000000	0.001428	0.000000
17	0.000000	0.003610	0.003610	0.003610
18	0.000988	0.000988	0.003211	0.001976
19	0.000797	0.000797	0.000683	0.001291
20	0.000269	0.000806	0.001343	0.000448
21	0.000000	0.000875	0.002186	0.001312
22	0.000983	0.000694	0.000751	0.000751

Table A.11: Overview of the probabilities of book items for all categories (given in the rows) having 21 or more bids (given in the columns).

Category	21 to 30 bids	31 to 40 bids	41 to 50 bids	51 or more bids
1	0.000000	0.000000	0.000335	0.000000
2	0.000754	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000115
4	0.000000	0.000000	0.000000	0.000000
5	0.000591	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000	0.000000
11	0.001116	0.000000	0.000000	0.000000
12	0.000000	0.000156	0.000000	0.000000
13	0.000000	0.000000	0.000510	0.000000
14	0.000000	0.000000	0.000178	0.000000
15	0.000000	0.000000	0.000000	0.000000
16	0.001428	0.000000	0.000000	0.000000
17	0.000000	0.000000	0.000000	0.000000
18	0.000000	0.000000	0.000247	0.000000
19	0.000000	0.000000	0.000418	0.000000
20	0.000358	0.000000	0.000000	0.000000
21	0.000000	0.000000	0.000000	0.000000
22	0.000173	0.000000	0.000000	0.000000

Table A.12: Overview of the probabilities of book items for all categories (given in the rows) with prices up to $4.00 (given in the columns).

Category	0.00 to 1.00	1.01 to 2.00	2.01 to 3.00	3.01 to 4.00
1	0.348310	0.166654	0.112922	0.076866
2	0.234263	0.191902	0.099524	0.109051
3	0.130725	0.166924	0.103796	0.085752
4	0.167331	0.121293	0.084108	0.100930
5	0.212462	0.205325	0.114741	0.108976
6	0.276529	0.182667	0.105103	0.084285
7	0.244309	0.201649	0.117584	0.103155
8	0.221018	0.118093	0.080715	0.098050
9	0.381329	0.087025	0.050633	0.037975
10	0.250000	0.096591	0.113636	0.125000
11	0.206428	0.138072	0.101856	0.081938
12	0.184201	0.170094	0.110105	0.093919
13	0.245265	0.241477	0.119555	0.097301
14	0.323060	0.180753	0.121580	0.089227
15	0.164268	0.179509	0.101609	0.137172
16	0.100659	0.135370	0.077751	0.114196
17	0.098446	0.150259	0.056995	0.111399
18	0.246017	0.144734	0.148769	0.109559
19	0.385819	0.191166	0.117430	0.077845
20	0.210760	0.181639	0.098466	0.092014
21	0.228517	0.132477	0.103623	0.078559
22	0.255159	0.254382	0.101731	0.098515

Table A.13: Overview of the probabilities of book items for all categories (given in the rows) with prices ranging from \$4.01 to \$8.00 (given in the columns).

Category	4.01 to 5.00	5.01 to 6.00	6.01 to 7.00	7.01 to 8.00
1	0.068251	0.039918	0.026253	0.025473
2	0.075536	0.044573	0.027731	0.034536
3	0.080234	0.050712	0.038020	0.042931
4	0.117751	0.067729	0.056662	0.048694
5	0.087977	0.059017	0.032116	0.030744
6	0.074114	0.044492	0.029562	0.026291
7	0.073400	0.047679	0.028410	0.031278
8	0.087216	0.060130	0.029252	0.033044
9	0.083861	0.036392	0.041139	0.050633
10	0.130682	0.079545	0.034091	0.028409
11	0.095971	0.035763	0.034405	0.038026
12	0.076398	0.050932	0.033484	0.034375
13	0.069839	0.044744	0.021780	0.026278
14	0.070070	0.037586	0.026428	0.025606
15	0.080440	0.060965	0.029636	0.049958
16	0.091982	0.078445	0.031586	0.049983
17	0.103627	0.077720	0.044041	0.056995
18	0.085558	0.057832	0.039727	0.028243
19	0.057004	0.032857	0.023640	0.019449
20	0.096204	0.049024	0.032682	0.035993
21	0.088037	0.052022	0.064027	0.061710
22	0.065713	0.049767	0.021895	0.031007

Table A.14: Overview of the probabilities of book items for all categories (given in the rows) with prices ranging from \$8.01 to \$30.00 (given in the columns). The first column contains the category, abbreviated by "C".

C	8.01 to 9.00	9.01 to 10.00	10.01 to 20.00	20.01 to 30.00
1	0.017118	0.035797	0.047234	0.013702
2	0.016672	0.048826	0.064137	0.019224
3	0.026708	0.052202	0.107604	0.046904
4	0.036299	0.076140	0.078353	0.019035
5	0.021136	0.043234	0.053253	0.014411
6	0.016893	0.039317	0.058589	0.022841
7	0.021061	0.032443	0.065782	0.015056
8	0.028169	0.070423	0.086132	0.029252
9	0.034810	0.104430	0.071203	0.009494
10	0.022727	0.039773	0.056818	0.011364
11	0.044364	0.095971	0.071073	0.022182
12	0.028955	0.058876	0.077957	0.029475
13	0.018939	0.036932	0.047585	0.012311
14	0.016036	0.035979	0.049099	0.012485
15	0.033870	0.055885	0.055038	0.024555
16	0.045817	0.091982	0.108990	0.032280
17	0.082902	0.054404	0.090674	0.044041
18	0.017277	0.032899	0.057107	0.014277
19	0.011525	0.024688	0.041828	0.009266
20	0.020112	0.052837	0.070980	0.025895
21	0.021904	0.046967	0.064869	0.029697
22	0.018224	0.029587	0.045881	0.011604

Table A.15: Overview of the probabilities of book items for all categories (given in the rows) with prices from $30.01 (given in the columns). The first column contains the category, abbreviated by "C".

C	30.01 to 40.00	40.01 to 50.00	50.01 to 100.00	100.00 or more
1	0.005830	0.005199	0.006276	0.004196
2	0.006975	0.006125	0.011398	0.009527
3	0.024004	0.013575	0.019589	0.010319
4	0.006197	0.005755	0.007525	0.006197
5	0.005078	0.003843	0.005764	0.001921
6	0.012788	0.008506	0.011301	0.006721
7	0.005736	0.004571	0.006722	0.001165
8	0.012459	0.014085	0.021668	0.010293
9	0.004747	0.001582	0.003165	0.001582
10	0.005682	0.000000	0.000000	0.005682
11	0.012675	0.009507	0.007696	0.004074
12	0.015591	0.010840	0.015963	0.008835
13	0.005208	0.003314	0.006155	0.003314
14	0.004766	0.002523	0.003420	0.001383
15	0.008467	0.009314	0.003387	0.005927
16	0.009719	0.011107	0.013190	0.006942
17	0.018135	0.005181	0.005181	0.000000
18	0.009311	0.003621	0.004138	0.000931
19	0.003291	0.001965	0.001948	0.000278
20	0.011565	0.007039	0.009134	0.005657
21	0.013269	0.005265	0.005687	0.003370
22	0.005976	0.003082	0.004690	0.002787

Table A.16: Overview of the number of items for all categories (given in the rows).

Category	Number of items
1	12, 244
2	2, 833
3	8, 951
4	9, 84
5	3, 514
6	7, 799
7	5, 456
8	911
9	262
10	103
11	911
12	6, 673
13	2, 051
14	23, 292
15	627
16	1, 450
17	271
18	4, 142
19	27, 172
20	11, 407
21	2343
22	18, 206

Appendix B

Attribute Domains & Predicate Ranges

In this appendix, we provide details about the realization of attribute domains in BoP (Appendix B.1). We also specify the ranges of the operands that are used in the predicates of subscriptions (Appendix B.2) and advertisements (Appendix B.3). The specifications we show in the following sections are applied in the experiments described in Chapter 5 and Chapter 8.

B.1 Domains and Data Types

In Table 3.1 we give an overview of the attribute domains of online auctions on eBay. Out of the 15 shown attributes, we chose the eight main attributes to be included in event messages.

We decided to map all of these attributes to the integer data type in BoP. This approach is straightforward to implement for enumerations and Boolean domains (attributes `Category`, `Format`, `Special Attribute`, `Condition`, and `Buy It Now`). Additionally, for numbers (attributes `Price` and `Bids`) and times (attribute `Ending Within`), this approach is uncomplicated: `Bids` is already defined as a natural number. `Price` involves two fractional digits; they are modeled as an integer by multiplying the original price by 100. For `Ending Within`, we use the number of minutes that is left in an auction for the internal representation.

Also for the attributes `Author` and `Title`, we decided to use the integer data type but not a string type. This approach only marginally affects attribute-value pairs in event messages and predicates in subscriptions: for event messages, the modeling as integer does not raise any validity questions because messages only contain a single value (in either representation). For subscriptions, our approach changes the semantics of predicates on `Title` (in Subscription Class 1 and Subscription Class 2, see Section 3.3.1, page 79) and `Author` (in Subscription Class 3, see Section 3.3.1) to exact string matching. In practice, this semantics for predicates is straightforward to realize, for example, by offering users a list of authors and book titles. Taking this approach, moreover, decreases the number of typographical errors by users.

By applying this simplification in our experiments, we can abstract from the exact values for the attributes `Author` and `Title` in event messages, and from the exact predicate specifications in subscriptions. Even if we used the string data type, the results of experiments do not become more realistic because we have not been able to derive the exact values for these attributes on eBay (see Section 3.2, page 72).

In general, the efficiency of predicate matching (see Section 4.3.1, page 107), and thus the efficiency of the overall filtering process in brokers, should decrease when using the string data type. For our comparative experiments, the analyzed algorithms utilize the same one-dimensional predicate index structures. The utilization of strings thus leads to the same increase in computational load for the compared algorithms. Hence our results do also hold when using strings as attribute domains.

For advertisements, we apply the same data types as for subscriptions in our experiments.

B.2 Predicates in Subscriptions

To derive a setting that is close to reality, we use subscriptions of the three introduced subscription classes (see Section 3.3) in our experiments. For the variable predicates of these subscriptions, we assign random but meaningful values according to the domains of the respective attributes. We give an overview of the ranges of the operands for these predicates in Table B.1. We consider five different distributions for the values of these operands:

- Uniform distribution
- Normal distribution (minimum value has the highest probability)
- Zipf distribution (minimum value has the highest probability)

Table B.1: Overview of the ranges of operands in the predicates of subscriptions.

Class	Predicate	Range
Class 1	p_1	Depends on B_{prop} (see Section 3.2.2, page 76)
Class 1	p_4	\$5.00 . . . \$10.00
Class 1	p_5	\$1.00 . . . \$5.00
Class 2	p_1	Depends on B_{prop} (see Section 3.2.2)
Class 2	p_6	\$12.00 . . . \$18.00
Class 2	p_7	\$8.00 . . . \$13.00
Class 2	p_{10}	\$10.00 . . . \$15.00
Class 2	p_{11}	\$5.00 . . . \$10.00
Class 3	p_1	22 categories (see Appendix A.1, page 301)
Class 3	p_2	Depends on B_{prop} and A_{prop} (see Section 3.2.2, page 76)

- Reversed normal distribution (maximum value has the highest probability)
- Reversed Zipf distribution (maximum value has the highest probability)

For the Zipf distributions with the k-th most popular value occurring $k^{-\alpha}$ as often as the most popular value, we set parameter $\alpha = 1$.

For the computation of the normal distributions, we set the mean value to $\mu = 0$ (representing the most popular value of the domain) and use a standard deviation σ of $\frac{1}{4}$ of the overall number of values in the domain. We map both sides of the bell curve (i.e., positive and negative values) to the same domain value. Values outside of the interval that we map to the normal distribution (interval $[-4\sigma, 4\sigma]$) are neglected (the number of such values is negligible due to the properties of the distribution). We truncate the derived values in order to map them to the discrete attribute domain.

B.3 Predicates in Advertisements

Similarly to subscriptions, when creating advertisements in experiments it is our goal to create a random but meaningful setting. That is, there should exist some overlap among advertisements and subscriptions in our experiments. We give an overview of the ranges of operands that are used within the predicates of advertisements in Table B.2. The distributions for the values of operands are the same as given in Appendix B.2.

Table B.2: Overview of the ranges of operands in the predicates of advertisements.

Class	Predicate	Range
Class 1	p_1	22 categories (see Appendix A.1, page 301)
Class 1	p_3	\$5.00 . . . \$30.00
Class 1	p_4	\$15.00 . . . \$30.00
Class 2	p_1	Depends on B_{prop} (see Section 3.2.2, page 76)
Class 2	p_3	\$5.00 . . . \$30.00
Class 2	p_4	\$15.00 . . . \$30.00
Class 3	p_1	Depends on B_{prop} and A_{prop}(see Section 3.2.2, page 76)
Class 3	p_3	\$3.00 . . . \$20.00
Class 3	p_4	\$8.00 . . . \$25.00
Class 4	p_1	Depends on B_{prop} (see Section 3.2.2)
Class 4	p_3	\$9.00 . . . \$20.00
Class 4	p_4	\$8.00 . . . EbayDollar18.00
Class 5	p_1	Depends on B_{prop} (see Section 3.2.2)
Class 5	p_5	\$9.00 . . . \$25.00
Class 5	p_6	\$9.00 . . . \$20.00
Class 5	p_9	\$8.00 . . . \$22.00
Class 5	p_{10}	\$3.00 . . . \$18.00
Class 6	p_2	\$3.00 . . . \$20.00
Class 6	p_3	\$8.00 . . . \$25.00
Class 7	p_2	\$9.00 . . . \$20.00
Class 7	p_3	\$8.00 . . . \$18.00
Class 8	p_4	\$9.00 . . . \$25.00
Class 8	p_5	\$9.00 . . . \$20.00
Class 8	p_8	\$8.00 . . . \$22.00
Class 8	p_9	\$3.00 . . . \$18.00

Bibliography

[AAGC04] J. Antollini, M. Antollini, P. Guerrero, and M. Cilia. Extending Rebeca to Support Concept-Based Addressing. In *Proceedings of the First Argentinean Symposium on Information Systems (ASIS '04)*, Cordoba, Argentina, September 23–24 2004. 25

[AJL02] G. Ashayer, H.-A. Jacobsen, and H. Leung. Predicate Matching and Subscription Matching in Publish/Subscribe Systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 539–548, Vienna, Austria, July 2–5 2002. IEEE Computer Society. 30, 37, 38, 39, 40, 41, 104, 116

[ASS^{+}99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching Events in a Content-Based Subscription System. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 53–61, Atlanta, USA, May 4–6 1999. ACM Press. 30, 32, 33, 34, 116

[BBC^{+}04] A. P. Buchmann, C. Bornhövd, M. Cilia, L. Fiege, F. C. Gärtner, C. Liebig, M. Meixner, and G. Mühl. DREAM: Distributed Reliable Event-Based Application Management. In *Web Dynamics*, pages 319–352. Springer-Verlag, 2004. 14, 25

[BBHM95] J. Bacon, J. Bates, R. Hayton, and K. Moody. Using Events to Build Distributed Applications. In *Second International Workshop on Services in Distributed and Networked Environments (SDNE '95)*, pages 148–155, Whistler, Canada, June 5–6 1995. IEEE Computer Society. 96

[BCM+99] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, Austin, USA, May 31–June 4 1999. IEEE Computer Society. 17

[BH04] S. Bittner and A. Hinze. Classification and Analysis of Distributed Event Filtering Algorithms. In *Proceedings of the 12th International Conference on Cooperative Information Systems (CoopIS 2004)*, pages 301–318, Agia Napa, Cyprus, October 25–29 2004. Springer-Verlag. 42, 43, 295

[BH05a] S. Bittner and A. Hinze. A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms. In *Proceedings of the 13th International Conference on Cooperative Information Systems (CoopIS 2005)*, pages 148–165, Agia Napa, Cyprus, October 31–November 4 2005. Springer-Verlag. 120

[BH05b] S. Bittner and A. Hinze. On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '05)*, pages 451–457, Columbus, USA, June 6–10 2005. IEEE Computer Society. 103, 104, 112, 243

[BH06a] S. Bittner and A. Hinze. Arbitrary Boolean Advertisements: The Final Step in Supporting the Boolean Publish/Subscribe Model. Technical Report 06/2006, Computer Science Department, The University of Waikato, June 2006. 206

[BH06b] S. Bittner and A. Hinze. Dimension-Based Subscription Pruning for Publish/Subscribe Systems. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '06)*, page 25, Lisbon, Portugal, July 4–7 2006. IEEE Computer Society. 162

[BH06c] S. Bittner and A. Hinze. Optimizing Pub/Sub Systems by Advertisement Pruning. In *Proceedings of the 8th Interna-*

tional Symposium on Distributed Objects and Applications (DOA 2006), pages 1503–1521, Montpellier, France, October 30–November 1 2006. Springer-Verlag. 217

[BH06d] S. Bittner and A. Hinze. Pruning Subscriptions in Distributed Publish/Subscribe Systems. In *Proceedings of the Twenty-Ninth Australasian Computer Science Conference (ACSC 2006)*, pages 197–206, Hobart, Australia, January 16–19 2006. ACS. 161

[BH07] S. Bittner and A. Hinze. The Arbitrary Boolean Publish/Subscribe Model: Making the Case. In *Proceedings of the Inaugural International Conference on Distributed Event-Based Systems (DEBS 2007)*, Toronto, Canada, June 20–22 2007. 14, 95

[Bit06] S. Bittner. Supporting Arbitrary Boolean Subscriptions in Distributed Publish/Subscribe Systems. In *Proceedings of the 3rd International Middleware Doctoral Symposium (MDS 2006)*, Melbourne, Australia, November 27–December 1 2006. ACM Press. 2

[Blo70] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970. 56

[Bon00] A. Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings of the 2nd International Workshop on Software and Performance (WOSP '00)*, pages 195–203, Ottawa, Canada, September 17–20 2000. ACM Press. 24, 27

[Bry86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986. 34, 116

[BW96] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs is NP-Complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996. 34

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, Harlow, 1999. 72

[CB02] M. Cilia and A. P. Buchmann. An Active Functionality Service For E-Business Applications. *ACM SIGMOD Record, Special Issue on Data Management Issues in Electronic Commerce*, 31(1):24–30, 2002. 65

[CBGM03] A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Query Merging: Improving Query Subscription Processing in a Multicast Environment. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):174–191, 2003. 54, 55, 197

[CCC$^+$01] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, pages 443–452, Toronto, Canada, May 12–19 2001. IEEE Computer Society. 23, 30, 34, 35, 36, 37, 39, 59, 97, 98, 100, 116, 117

[CF03] R. Chand and P. A. Felber. A Scalable Protocol for Content-Based Routing in Overlay Networks. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA 2003)*, pages 123–130, Cambridge, USA, April 16–18 2003. IEEE Computer Society. 17, 51, 197

[CFMP04] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the Reconfiguration Overhead in Content-Based Publish-Subscribe. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*, pages 1134–1140, Nicosia, Cyprus, March 14–17 2004. ACM Press. 18

[CKKM00] Z. Chen, N. Koudas, F. Korn, and S. Muthukrishnan. Selectively Estimation For Boolean Queries. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2000)*, pages 216–225, Dallas, USA, May 15–17 2000. ACM Press. 198

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, 2001. 194

[CMPC03] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Epidemic Algorithms for Reliable Content-Based Publish-Subscribe: An Evaluation. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS '03)*, pages 552–561, Providence, USA, May 19–22 2003. IEEE Computer Society. 23, 25

[CNF01] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering (TSE)*, 27(9):927–850, 2001. 17, 91

[CRW99] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Interfaces and Algorithms for a Wide-Area Event Notification Service. Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, October 1999. revised May 2000. 230

[CRW00] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC '00)*, pages 219–227, Portland, USA, July 16–19 2000. ACM Press. 23, 25

[CRW01] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001. 17, 32, 43, 44, 50, 51, 197, 230, 231

[CRW04] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A Routing Scheme for Content-Based Networking. In *Proceedings of the 23rd IEEE Conference on Computer Communications (INFOCOM 2004)*, Hong Kong, China, March 7–11 2004. IEEE Computer Society. 17

[CS04] F. Cao and J. P. Singh. Efficient Event Routing in Content-Based Publish/Subscribe Service Network. In *Proceedings of the 23rd IEEE Conference on Computer Communications*

(INFOCOM 2004), Hong Kong, China, March 7–11 2004. IEEE Computer Society. 17, 25

[CW03] A. Carzaniga and A. L. Wolf. Forwarding in a Content-Based Network. In *Proceedings of the 2003 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, pages 163–174, Karlsruhe, Germany, March 24–26 2003. ACM Press. 5, 69, 113

[Dag07] Event Processing. Dagstuhl Seminar 07191, 2007. 296

[DEB06] DEBS Participants. Discussion at the 6th International Workshop on Distributed Event-Based Systems (DEBS '06), July 3 2006. 296

[DRW06] L. Duboc, D. S. Rosenblum, and T. Wicks. A Framework for Modelling and Analysis of Software Systems Scalability. In *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*, pages 949–952, Shanghai, China, May 20–28 2006. ACM Press. 24

[EFGH02] P. T. Eugster, P. Felber, R. Guerraoui, and S. B. Handurukande. Event Systems: How to Have Your Cake and Eat It Too. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 625–632, Vienna, Austria, July 2–5 2002. IEEE Computer Society. 25, 198

[EFGK03] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003. 14, 25

[Eug01] P. T. Eugster. *Type-Based Publish/Subscribe.* PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), December 2001. 96

[FJL$^+$01] F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *Proceedings of the 2001*

ACM SIGMOD International Conference on Management of Data (SIGMOD 2001), pages 115–126, Santa Barbara, USA, May 21–24 2001. ACM Press. 23, 30, 32, 33, 37, 38, 116, 128

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, 34(3):596–615, 1987. 194

[GD98] B. Genet and G. Dobbie. Is Semantic Optimisation Worthwhile? In *Proceedings of the 21st Australasian Computer Science Conference (ACSC 1998)*, pages 245–256, Perth, Australia, February 1998. Springer-Verlag. 102

[GFGRAGC98] E. Giménez-Funes, L. Godo, J. Rodríguez-Aguilar, and P. Garcia-Calvés. Designing Bidding Strategies for Trading Agents in Electronic Auctions. In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS 1998)*, pages 136–143, Paris, France, July 3–7 1998. IEEE Computer Society. 68

[GR03] M. Guimarães and L. Rodrigues. A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA 2003)*, pages 67–74, Cambridge, USA, April 16–18 2003. IEEE Computer Society. 164

[GS95] J. Gough and G. Smith. Efficient Recognition of Events in a Distributed System. In *Proceedings of the 18th Australasian Computer Science Conference (ACSC 1995)*, Adelaide, Australia, February 1–3 1995. ACS. 30, 32, 33, 116

[GT98] A. Geppert and Dimitrios Tombros. Event-based Distributed Workflow Execution with EVE. In *Proceddings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 427–442, The Lake District, UK, September 15–18 1998. Springer-Verlag. 91

[Hah01] J. Hahn. The Dynamics of Mass Online Marketplaces: A Case Study of an Online Auction. In *Proceedings of the SIG-CHI*

Conference on Human Factors in Computing Systems, pages 317–324, Seattle, USA, March 31–April 5 2001. ACM Press. 77

[Haw01] S. W. Hawking. *The Universe in a Nutshell*. Bantam, New York, 2001. 2

[HB02] A. Hinze and S. Bittner. Efficient Distribution-based Event Filtering. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 525–532, Vienna, Austria, July 2–5 2002. IEEE Computer Society. 295

[HCH$^+$99] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. B. Park, and A. Vernon. Scalable Trigger Processing. In *Proceedings of the 15th International Conference on Data Engineering (ICDE '99)*, pages 266–275, Sydney, Australia, March 23–26 1999. IEEE Computer Society. 30, 37

[HCRW04] C. P. Hall, A. Carzaniga, J. Rose, and A. L. Wolf. A Content-Based Networking Protocol For Sensor Networks. Technical Report CU-CS-979-04, Department of Computer Science, University of Colorado, August 2004. 22

[Hei05] D. Heimbigner. Expressive and Efficient Peer-to-Peer Queries. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, USA, January 3–6 2005. IEEE Computer Society. 230

[HGM01] Y. Huang and H. Garcia-Molina. Publish/Subscribe in a Mobile Environment. In *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '01)*, pages 27–34, Santa Barbara, USA, May 20 2001. ACM Press. 14

[Hil90] M. D. Hill. What is Scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990. 24

[Hin03] A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universität Berlin, Institute of Computer Science, July 2003. 5, 17, 230

[JH05a] D. Jung and A. Hinze. A Mobile Alerting System for the Support of Patients with Chronic Conditions. In *Proceedings of the First European Conference on Mobile Government (EURO mGOV 2005)*, pages 264–274, Brighton, UK, July 10–12 2005. 91

[JH05b] D. Jung and A. Hinze. Capturing Context in Collaborative Profiles. In *Proceedings of On the Move to Meaningful Internet Systems 2005: OTM Workshops, Workshop on Context-Aware Mobile Systems (CAMS 2005)*, pages 152–155, Agia Napa, Cyprus, October 30–31 2005. 91

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16(2):111–152, 1984. 6, 59

[JMS99] S. Jones, S. McInnes, and M. S. Staveley. A Graphical User Interface for Boolean Query Specification. *International Journal on Digital Libraries (IJDL)*, 2(2–3):207–223, 1999. 72

[JT92] R. P. Jacobi and A.-M. Trullemans. Generating Prime and Irredundant Covers for Binary Decision Diagrams. In *Proceedings of the 3rd European Conference on Design Automation*, pages 104–108, Brussels, Belgium, March 16–19 1992. IEEE Computer Society. 35, 100, 116

[JTS04] N. Joshi, K. Thakore, and S. Y. W. Su. IntelliBid: An Event-Trigger-Rule-Based Auction System over the Internet. *World Wide Web Journal (W3J)*, 7(2):181–210, 2004. 69

[Jun07] D. Jung. Realisation and Representation of Collaborative Profiles for Alerting in Health Care. In *Graduate Workshop of the Eighth Annual ACM SIGCHI-NZ Conference on Human-Computer Interaction (CHINZ 2007)*, Hamilton, New Zealand, July 1–4 2007. 72

[KMPS94] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing Disjunctive Queries with Expensive Predicates. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD 1994)*, pages 336–347, Minneapolis, USA, May 24–27 1994. ACM Press. 59

[LCB99] C. Liebig, M. Cilia, and A. Buchmann. Event Composition in Time-Dependent Distributed Systems. In *Proceedings of the 4th International Conference on Cooperative Information Systems (CoopIS 1999)*, pages 70–78, Edinburgh, UK, September 2–4 1999. IEEE Computer Society. 23

[Leh05] W. Lehner. Data Management Support for Notification Services. In *Data Management in a Connected World: Essays Dedicated to Hartmut Wedekind on the Occasion of His 70th Birthday*, pages 111–136. Springer-Verlag, June 2005. 14

[LHJ05] G. Li, S. Hou, and H.-A. Jacobsen. A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems based on Modified Binary Decision Diagrams. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 447–457, Columbus, USA, June 6–10 2005. IEEE Computer Society. 17, 25, 31, 35, 36, 37, 51, 53, 54, 100, 116, 117, 162, 197, 230, 231, 232

[LJ03] H. K. Y. Leung and H.-A. Jacobsen. Efficient Matching for State-Persistent Publish/Subscribe Systems. In *Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '03)*, pages 182–196, Toronto, Canada, October 6–9 2003. IBM Press. 14, 25

[LJ04] H. Liu and H.-A. Jacobsen. Modeling Uncertainties in Publish/Subscribe Systems. In *Proceedings of the 20th International Conference on Data Engineering (ICDE '04)*, pages 510–522, Boston, USA, March 30–April 02 2004. IEEE Computer Society. 69

[LJ05] G. Li and H.-A. Jacobsen. Composite Subscriptions in Content-Based Publish/Subscribe Systems. In *Proceedings* *of the 6th ACM/IFIP/USENIX International Conference on Middleware (Middleware '05)*, pages 249–269, Grenoble, France, November 28–December 2 2005. Springer-Verlag. 91, 92

[LSB06] C. Lumezanu, N. Spring, and B. Bhattacharjee. Decentralized Message Ordering for Publish/Subscribe Systems. In *Proceedings of the 7th ACM/IFIP/USENIX International Conference on Middleware (Middleware '06)*, pages 62–179, Melbourne, Australia, November 27–December 1 2006. ACM Press. 23

[LV03] P. Lyman and H. R. Varian. How Much Information, 2003. Retrieved from `http://www.sims.berkeley.edu/how-much-info-2003` on 16 April 2007. 1

[LW04] K. M. Lochner and M. P. Wellman. Rule-Based Specification of Auction Mechanisms. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*, pages 818–825, New York, USA, July 19–23 2004. IEEE Computer Society. 69

[Men97] E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall/CRC, Boca Raton, 1997. 59

[MF01] G. Mühl and L. Fiege. Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs. *IEEE Distributed Systems Online (DSOnline)*, 2(7), 2001. 4, 50, 51, 53, 54, 197

[MF05] A. Michlmayr and P. Fenkam. Integrating Distributed Object Transactions with Wide-Area Content-Based Publish/Subscribe Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '05)*, pages 398–403, Columbus, USA, June 6–10 2005. IEEE Computer Society. 23

[MFB02] G. Mühl, L. Fiege, and A. Buchmann. Filter Similarities in Content-Based Publish/Subscribe Systems. In *Proceedings of the International Conference on Architecture of Computing*

Systems (ARCS '02), pages 224–238, Karlsruhe, Germany, April 8–12 2002. Springer-Verlag. 59

[MFP06] G. Mühl, L. Fiege, and P. R. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag, Berlin and Heidelberg, 2006. 3, 31, 33, 34, 37, 42

[MG85] J. MacKinley and M. R. Genesereth. Expressiveness and Language Choice. *Data Knowledge Engineering (DKE)*, 1(1):17–29, 1985. 25

[MK60] M. E. Maron and J. L. Kuhns. On Relevance, Probabilistic Indexing and Information Retrieval. *Journal of the ACM*, 7(3):216–244, 1960. 72

[Müh01] G. Mühl. Generic Constraints for Content-Based Publish/Subscribe Systems. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS 2001)*, pages 211–225, Trento, Italy, September 5–7 2001. Springer-Verlag. 153

[Müh02] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische Universität Darmstadt, September 2002. 17, 25, 43, 44, 46, 51, 52, 54, 60, 69, 116, 204, 205, 213, 230, 231, 261

[OJPA06] A. M. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient Probabilistic Subsumption Checking for Content-based Publish/Subscribe Systems. In *Proceedings of the 7th ACM/IFIP/USENIX International Conference on Middleware (Middleware '06)*, Melbourne, Australia, Nov 27–Dec 1 2006. Springer-Verlag. 51, 197

[PB02] P. R. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 611–618, Vienna, Austria, July 2–5 2002. IEEE Computer Society. 45

[PCM03] G. P. Picco, G. Cugola, and A. L. Murphy. Efficient Content-Based Event Dispatching in the Presence of Topological Reconfiguration. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS '03)*, pages 234–243, Rhode Island, USA, May 19–22 2003. IEEE Computer Society. 23, 25

[PFLS00] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient Matching for Web-Based Publish/Subscribe Systems. In *Proceedings of the 7th International Conference on Cooperative Information Systems (CoopIS 2000)*, pages 162–173, Eilat, Israel, September 6–8 2000. Springer-Verlag. 31

[PI97] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997)*, pages 486–495, Athens, Greece, August 25–29 1997. Morgan Kaufmann. 198

[Pie04] P. R. Pietzuch. *Hermes: A Scalable Event-Based Middleware.* PhD thesis, University of Cambrigde, Queens' College, February 2004. 4, 23, 45, 46, 51, 52, 96, 230, 231, 261

[PSB04] P. R. Pietzuch, B. Shand, and J. Bacon. Composite Event Detection as a Generic Middleware Extension. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks*, 18(1):44–55, 2004. 23

[RDJ02] W. Rjaibi, K. R. Dittrich, and D. Jaepel. Event Matching in Symmetric Subscription Systems. In *Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '02)*, Toronto, Canada, September 30 – October 2 2002. IBM Press. 32, 33

[RKCD01] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Proceedings of the 3rd International Workshop on Networked Group Communications (NGC 2001)*, pages 30–43, London, UK, November 7–9 2001. Springer-Verlag. 46

[Ros04] K. A. Ross. Selection Conditions in Main Memory. *ACM Transactions on Database Systems (TODS)*, 29(1):132–161, 2004. 72

[Ros05] D. Rosenblum. Content-Based Publish/Subscribe: A Re-Assessment, 2005. Keynote at the 7th International Symposium on Distributed Objects and Applications (DOA 2005). 296

[RR06] C. Raiciu and D. S. Rosenblum. Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures. In *Proceedings of the Second IEEE/CreatNet International Conference on Security and Privacy in Communication Networks (SecureComm 2006),*, Baltimore, USA, August 28–September 1 2006. IEEE Computer Society. 22

[RWG01] D. M. Reeves, M. P. Wellman, and B. N. Grosof. Automated Negotiation from Declarative Contract Descriptions. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents 2001)*, pages 51–58, Montreal, Canada, May 28–June 1 2001. ACM Press. 67

[SA97] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the Australian UNIX and Open Systems User Group Conference (AUUG97)*, Brisbane, Australia, September 3–5 1997. 30, 31, 32, 34, 39, 116

[SAB+00] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based Routing with Elvin4. In *Proceedings of Australian UNIX and Open Systems User Group Conference (AUUG2K)*, Canberra, Australia, June 25–30 2000. 30, 31, 32, 34, 39

[Sal68] G. Salton. *Automatic Information Organization and Retrieval.* McGraw-Hill, New York, 1968. 4

[Sil86] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall/CRC, Boca Raton, 1986. 73

[SL95] A. Stepanov and M. Lee. The Standard Template Library. Technical Report 95-11 (R.1), HP Laboratories, November 1995. 138

[Sri93] D. Srivastava. Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):315–343, 1993. 51

[Ste86] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application*. John Wiley & Sons Inc, Chichester, 1986. 187

[TAJ03] D. Tam, R. Azimi, and H.-A. Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *Proceedings of First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P 2003)*, pages 138–152, Berlin, Germany, September 7–8 2003. Springer-Verlag. 25

[Tar06] S. Tarkoma. Preventing Spam in Publish/Subscribe. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '06)*, page 21, Lisbon, Portugal, July 4–7 2006. IEEE Computer Society. 22

[TE02] P. Triantafillou and A. Economides. Subscription Summaries for Scalability and Efficiency in Publish/Subscribe Systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 619–624, Vienna, Austria, July 2–5 2002. IEEE Computer Society. 56, 197

[TE04] P. Triantafillou and A. Economides. Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS '04)*, pages 562–571, Tokyo, Japan, March 24–26 2004. IEEE Computer Society. 56, 57, 197

[TKD04] C. Tryfonopoulos, M. Koubarakis, and Y. Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2004)*, pages 313–320, Sheffield, UK, July 25–29 2004. ACM Press. 31

[WCEW02] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems. In *Proceedings of the 35th Hawaii International Conference on System Science (HICSS-35)*, pages 3940–3947, Big Island, USA, January 7–10 2002. IEEE Computer Society. 22

[WK05] B. Wang and M. Kitsuregawa. Dimension Transform Based Efficient Event Filtering for Symmetric Publish/Subscribe System. In *Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA 2005)*, pages 786–796, Copenhagen, Denmark, August 22–26 2005. Springer-Verlag. 32, 33

[WMB99] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann, San Francisco, 1999. 67

[WQV$^+$04] Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson. Summary-based Routing for Content-based Event Distribution Networks. *ACM SIGCOMM Computer Communication Review*, 34(5):59–74, 2004. 56, 57, 197, 231

[WWW01] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A Parametrization of the Auction Design Space. *Games and Economic Behavior*, 35(1–2):204–238, 2001. 66

[YB05] E. Yoneki and J. Bacon. Distributed Multicast Grouping for Publish/Subscribe over Mobile Ad Hoc Networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC' 2005)*, pages 2293–2299, New Orleans, USA, March 13–17 2005. IEEE Computer Society. 56

[YGM94] T. W. Yan and H. García-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems (TODS)*, 19(2):332–364, 1994. 4, 5, 30, 38, 39, 40, 116

[YGM99] T. W. Yan and H. García-Molina. The SIFT Information Dissemination System. *ACM Transactions on Database Systems (TODS)*, 24(4):529–565, 1999. 5, 31

[YS93] D. Young and B. Shneiderman. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. *Journal of the American Society for Information Science*, 44(6):327–339, 1993. 72, 82

[ZZJ$^+$01] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Widearea Data Dissemination. In *Proceddings of the 11th International Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, pages 11–20, Port Jefferson, USA, June 25–27 2001. ACM Press. 46