Thesis for the Degree of Habilitation

PRINCIPLES OF BUILDING SCALABLE AND ROBUST EVENT-BASED SYSTEMS

DR. BORIS KOLDEHOFE

Distributed network-centric adaptation of event processing and publish/subscribe

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Electrical Engineering and Information Technology
Technische Universität Darmstadt

March 2019

Principles of building scalable and robust event-based systems
—Distributed network-centric adaptation of event processing and publish/subscribe
Dr. Boris Koldehofe

Habiliationsschrift an der Technischen Universität Darmstadt
im Fach Kommunikation und Verteilte Systeme

Technische Universität Darmstadt
Department of Electrical Engineering and Information Technology
Fraunhoferstraße 4
64283 Darmstadt
Germany

Darmstadt, Germany, March 2019

ABSTRACT

Event-based systems are of tremendous importance for a wide range of distributed applications interacting with physical processes, e. g., traffic management, financial services, manufacturing processes, or health services. Event-based systems support to monitor, analyze events of interest efficiently. Therefore, they enable distributed applications to respond to detected events in the form of appropriate actions. Event-based systems provide as part of the *publish/subscribe* paradigm, mechanisms for the scalable integration of a variety of information sources, e. g., dedicated sensor networks, mobile devices, or cameras. In addition, event-based systems allow as part of the *event processing paradigm* to detect correlations between events from distinct information sources. Event-based systems ensure two important forms of decoupling of importance building scalable distributed applications. Decoupling producers of information and consumers of information by ensuring that neither producers need to keep state on the interested consumers nor consumers need to know the producers of information, is a key principle for scalable communications. Furthermore, a step-wise correlation from primary events to events of importance for distributed applications is an enabler to specify distributed applications independent from the underlying sensor infrastructure at hand.

In this thesis, we present and discuss principles of building scalable and robust event-based systems. On the one hand, this requires distributed mechanisms to fulfill a wide spectrum of distinct application requirements, e. g., being bandwidth efficient and providing events with low end-to-end latency. On the other hand, the underlying mechanisms for event-based systems need to deal with many levels of dynamics, e. g., dynamics in the rate at which events are produced, dynamics in the interest of producers and consumers, mobility of consumer and producer, failures and changing security privileges to access events. In the context of mechanisms for event distribution, operator execution, operator migration, operator recovery and secure access to events, we highlight problems in the scalable and robust design of those mechanisms. We give an overview on related work in the field and present in a tutorial manner the ideas of six own contributions for realizing distributed event-based systems.

ACKNOWLEDGMENTS

The presented ideas and contributions as part of this habilitation thesis are the outcome of many joint research works and co-supervised Ph.D. and master theses at the University of Stuttgart and Technical University of Darmstadt. Therefore, I thank a lot all (current and former) members of the adaptive communication systems group (AKS) in Stuttgart and the adaptive overlay communications group (AOC) in Darmstadt for all the great thoughts and work we share together. For the selected contributions I owe special thanks to M. Adnan Tariq, Beate Ottenwälder, Björn Schilling, Gerald G. Koch, Sukanya Bowmik and Ruben Mayer for many years of joint cooperation and enthusiastically and critically developing, refining, analyzing and evaluating the proposed principles and approaches. Furthermore, I would like to thank Kurt Rothermel and Ralf Steinmetz for sharing excellent and inspiring research environments as well as for many intensive and helpful research discussions on and beyond the topic. Thanks a lot to the BW-Stiftung and the German Science Foundation (DFG) for having offered research grants that supperted the work, i. e., as part of the Spitzenforschung Baden Württemberg, the SpoVNet project and the subproject C2 of the DFG Collaborative Research Centre 1053 MAKI—Multi-Mechanisms Adaptation for the Future Internet.

Many thanks to Wolfgang Effelsberg for providing detailed feedback on the draft of this thesis and encouragement. Furthermore, I would like to thank Umakishore Ramachandran for many exciting research discussions along the topics of mobile complex event processing. Last, but not least thanks a lot to all friends, family members, colleagues, students, guests and reviewers for all support and many interesting pointers that contributed to my excitement and pleasure working in this problem space.

CONTENTS

ACRONYMS

IOT Internet of Things 1

MCEP Mobile Complex Event Processing 38

ONF Open Network Foundation 20

QOS Quality of Service 11

RFID Radio Frequency Identification 1

SDN Software-defined Networking 3

TCAM Ternary Content Addressable Memory 18

TCEP Transition capable complex event processing 39

1 INTRODUCTION

1.1 EVENT-BASED COMMUNICATIONS

Detecting situational changes is important for many distributed applications

The ability to autonomously adapt to changing situational conditions is inherent in a broad range of application domains including the Internet of Things (IoT). Logistics, manufacturing, financial services, health care, traffic monitoring, or energy management, are just a few examples for processes which need to adapt to changes such as traffic conditions, the status of a production step, or the health status of a patient. Being aware of such changes, these processes can adapt by triggering actions like changing the route of a vehicle, modifying the manufacturing process or simply triggering an alarm.

As part of this adaptive process, applications can build on an increasing variety and tremendous number of information sources, e. g., billions of IoT devices comprising sensors, mobile phones, cameras and Radio Frequency Identification (RFID) readers, but also web feeds or databases, to understand and detect when situational changes are happening. Furthermore, applications in these domains are highly distributed and comprise many entities, e. g., the navigator of a vehicle, the controller of a machine, or the monitoring unit of a patient, that perform concurrent and often independent local adaptations.

IoT resources provide huge variety and number of information sources

In order to avoid performance bottlenecks and allow for a continuous evolution of applications and their underlying infrastructure, ensuring a *decoupling* of interacting entities is a key design principle [23]. A decoupling of detecting situational changes from the actual application logic allows for modifications on the way detecting situational changes without the need to modify the application logic itself. Moreover, a decoupling in the communication of producers and consumers of information releases the application knowing and directly interacting with every information source. This a common and important bottleneck that must be overcome in a wide range of distributed applications. Ensuring a decoupling, helps and is essential that many entities can benefit from and participate in the process of detecting situational changes.

Decoupling is needed for scalability and evolvability

Event-based communications has evolved as the key paradigm in realizing a decoupling in the process of detecting situational changes and notifying applications about their happenings. In an event-based system, i. e., a system for realizing event-based communications, the happening of situational change is encapsulated by the notion of an *event*. Applications interact with an event-based system by subscribing to events in the role of an *event consumer* and producing events

Event-based communications is a paradigm ensuring a decoupling in detecting situational changes

in the role of an *event producer*. An important property of an event-based system is that producers and consumers of information can interact without explicit knowledge about each other. Furthermore, the event-based system can transform low-level events (corresponding to primary sensor data) to complex events (corresponding to situational changes) of interest to the application. In event-based systems these properties are addressed by (complex) event processing and publish/-subscribe systems.

Event-based based communications must meet in a scalable and reliable way application requirements

The decoupling of event-based communications is widely recognized as a key ingredient for building scalable distributed applications. Nevertheless, a decoupling alone is not sufficient to ensure that event-based systems can meet a wide range of important application requirements like low latency delivery, mobility of users, reliability in the presence of failures. In order to fulfill such requirements, event-based systems have to account for many design decisions. These comprise which processing entities, e.g. dedicated servers, mobile devices participate in detecting a situation, and how and to which entities events should be forwarded. Furthermore, event-based systems have to combine forwarding mechanisms for ensuring reliability and security, dealing with dynamic changes such as dynamic subscriptions and unsubscriptions to events, load variations of events, and failures.

Therefore, an understanding on design and adaption principles is highly important for event-based systems to meet distinct requirements in a wide range of conditions.

1.2 THESIS GOALS AND PROBLEMS

Publish/subscribe and event processing are central paradigms for establishing a decoupling between producers and consumers

As part of this thesis, we aim to offer the reader an in-depth understanding in design decision, mechanisms, and methods for building scalable and robust event-based systems that can meet a variety of distinct application requirements. Most prominent application requirements covered in this thesis comprise bandwidth efficiency, low latency, mobility, reliability, and security. We concentrate on two central paradigms in realizing a decoupling between event producers and event consumers. *Publish/subscribe*, offers mechanisms and methods to route events from producers to interested consumers. *Event processing* is a a paradigm to accomplish the transformation of primary events to complex events. For these paradigms, we introduce key problems, corresponding related work and contributions of our own in the field. In the following we give an overview on the key problems detailed in subsequent chapters of the the thesis.

Publish/subscribe problems

Problem 1.1: scalable routing and topology management

In the context of *publish/subscribe* a broker network routes events from event producers to event consumers. A key challenge for broker networks is that the interest of consumers can vary significantly and frequently change. Therefore, in this thesis we aim to analyze and

introduce mechanisms that accomplish the broker functionality in a highly decentralized way in order to support a large number of producers and consumers and yet meet specific application requirements. This requires appropriate routing algorithms and topology management mechanisms to ensure brokers can route events from producers to consumers according to application requirements at hand (*Problem 1.1*).

Problem 1.2: multiple application requirements

A central application requirement —being bandwidth efficient— depends on the ability of consumers to express subscriptions to events in a fine-grained manner. However, other additional but conflicting application requirements, e. g., end-to-end latency, can suffer from very fine-grained specifications of subscriptions. Therefore, we study as part of the thesis mechanisms that allow to adapt and control the granularity at which subscriptions are specified in order to account for multiple application requirements (*Problem 1.2*).

Problem 1.3: removing bottlenecks by hardware acceleration

Furthermore, we aim to understand and propose methods to remove important performance bottlenecks by accounting for the underlying communication infrastructure (*Problem 1.3*). As part of this thesis we study such performance bottlenecks and analyze how they can be removed by building on more flexible mechanisms in controlling communication networks as provided by Software-defined Networking (SDN).

Problem 1.4: scalable key management

Another important application requirement which stays in conflict to both latency-sensitive and bandwidth-efficient routing of events is ensuring the confidentiality of events a highly distributed compute and communication infrastructure. This requires both appropriate mechanisms to encrypt and verify the authenticity of events and a corresponding scalable key management (*Problem 1.4*).

Event processing problems

In the context of *event processing* the brokers also detect correlations, i. e., pattern occurring in the incoming event streams. Whenever an event pattern is detected the broker produces in turn complex events. The logic in detecting event patterns, i. e., the algorithm to detect a complex event, is commonly encapsulated in form of an *operator*. Operators can be placed and executed on brokers in the network. Moreover, multiple dependent correlations can be modeled by an operator graph.

Problem 2.1: Parallelization of operators

In order to ensure stable end-to-end latency, the operator's speed in processing events needs to catch up with the arrival rate of events. Since in event-based systems the arrival rate of events can vary heavily, the rates can easily exceed the processing capacity of a single broker. As part of the thesis, we deal with the dynamic scaling of the operator's processing speed by parallelizing the operator logic. We discuss methods for the parallel execution of operators and adaptive algorithms in changing the parallelization degree to meet predictable end-to-end latency bounds (*Problem 2.1*).

Problem 2.2: Operator migrations

The placement of operators of an operator graph heavily influences the ability of event processing to meet application requirements. Especially mobility of consumer can frequently change the optimal placement and requires frequent migrations. In the cause of mobility also the context of the user and its interest can change. This requires operators to dynamically select events from producers at distinct locations. As part of the operator migration problem, we look into the problem of efficiently moving the operator state and provide mobility-aware mechanisms for event specification and processing (*Problem 2.2*).

Problem 2.3: Reliability

In the presence of failures, e. g., a crash of a broker, event processing systems should still be able to detect and deliver events of interest of consumers. As part of the reliable event processing problem (*Problem 2.3*, we investigate mechanisms that allow to efficiently recover from failures and ensure at the same time an acceptable run-time overhead. Furthermore, we aim to ensure the consistent processing of events, e. g., the mechanisms should avoid delivering events which would not be detected in failure free execution (no false positives) and ensure all detectable events will eventually be detected (no false negative detection).

1.3 CONTRIBUTIONS

In addressing Problems 1.1–1.4 and Problems 2.1–2.3, the cumulative habilitation thesis contributes with six published findings (F1–F6). Publications F1–F3 contribute to mechanisms for publish/subscribe and publications F4–F6 to mechanisms for event processing.

F1: Muhammad Adnan Tariq, Boris Koldehofe, Gerald Georg Koch, Imran Khan, and Kurt Rothermel. "Meeting subscriber-defined QoS constraints in publish/subscribe systems." In: *Concurrency and Computation: Practice and Experience*. Wiley, 2011, 23.11, pp. 2140–2153. DOI: 10.1002/cpe.1751

F2: Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler, and Kurt Rothermel. "High Performance Publish/Subscribe Middleware in Software-Defined Networks." In: *IEEE/ACM Transactions on Networking*. IEEE, 2017, 25.3, pp. 1501–1516. DOI: 10.1109/tnet.2016.2632970

F3: Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. "Securing Broker-Less Publish/Subscribe Systems Using Identity-Based Encryption." In: *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2014, 25.2, pp. 518–528. DOI: 10.1109/tpds.2013.256

F4: Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. "Predictable Low-Latency Event Detection With Parallel Complex

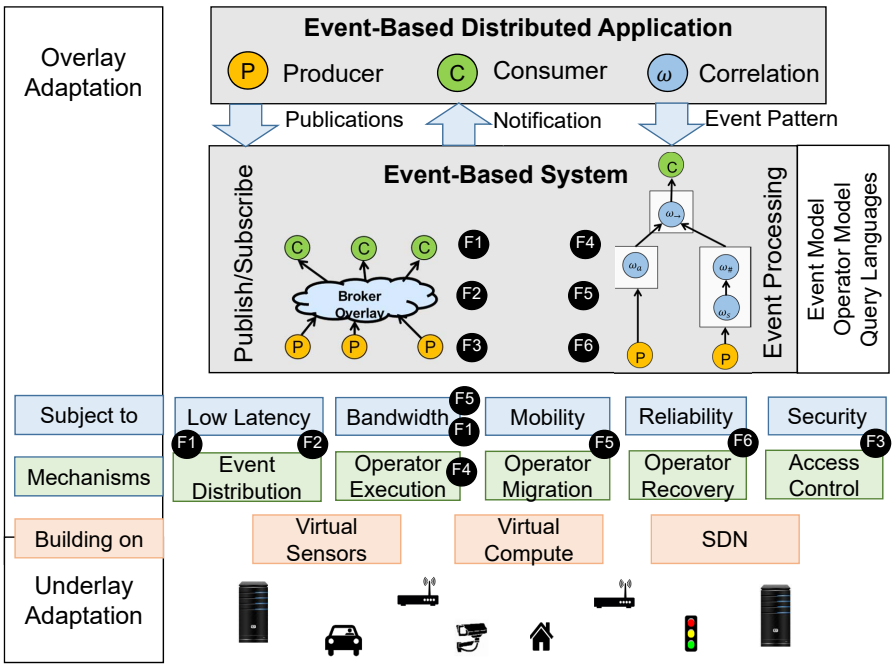


Figure 1.1: Overview of Contributions.

Event Processing." In: *IEEE Internet of Things Journal*. IEEE, 2015, 2.4, pp. 274–286. DOI: 10.1109/jiot.2015.2397316

F5: Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. "MCEP: A Mobility-Aware Complex Event Processing System." In: *ACM Transactions on Internet Technology*. ACM Press, 2014, 14.1, pp. 1–24. DOI: 10.1145/2633688

F6: Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. "Rollback-recovery without checkpoints in distributed event processing systems." In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 27–38. DOI: 10.1145/2488222.2488259

Figure 1.1 gives a more detailed overview of the contributions from the perspective of a distributed application comprising event producers, that publish with the publish/subscribe paradigm and corresponding mechanisms (F1–F3) events. Event consumers are asynchronously notified about events based on their subscriptions. In addition the distributed application can specify and efficiently detect event patterns (correlations) by building on appropriate mechanisms of event processing (F4–F6). The mechanisms for event processing are adapted subject to application requirements such as latency (F1, F2), bandwidth efficiency (F1, F5), mobility support (F3), reliability (F5) and security (F6). Corresponding mechanisms, i. e., mechanisms for

event distribution (F1,F2), operator execution (F4), operator migration (F5), operator recovery (F6), and access control (F2) are realized over a distributed infrastructure comprising things and user devices, servers, and network elements. The mechanisms for realizing the mechanisms of an event-based system can build on abstractions for virtual compute, software-defined networking, and virtual sensors.

1.4 THESIS STRUCTURE

The remainder of the habilitation thesis is structured as follows: In Chapter 2 we first introduce necessary models and notation, we will use in the subsequent chapters. Chapter 3–Chapter 7 deal with one of five specific mechanisms

1. event distribution
2. operator execution
3. operator migration
4. operator recovery
5. access control

illustrated in Figure 1.1. Each of these chapters follows the same structure, namely introducing into the research problems, discusses related work, and presents in a tutorial style solutions based on own contributions (F1–F6). Finally, Chapter 8 gives concluding remarks and an outlook on future work in the field.

2 BACKGROUND EVENT-BASED COMMUNICATIONS

In this chapter, we introduce the basic models we build on realizing distributed event-based systems. In particular, we provide a basic understanding of events and their representation. We introduce the basic system model for building distributed applications over event-based systems. Furthermore, we state a geometric representation for subscriptions of consumers and advertisements of producers, introduce models for the execution of operators and their distributed execution in the form of the operator graph model.

2.1 EVENT MODEL

As noted by Luckam [68], an *event* in a human context corresponds to the happening of something. Transferred to IT-systems, an event corresponds to any piece of information an IT-system may act upon, for instance adjusting a cooling system once a significant rise in temperature has been observed, or changing the traffic flow once a traffic jam has been detected. In order to be able to process events, an event needs to have some general representation that allows accessing important properties of the event and describes its nature. A common representation that has been successfully adapted in distributed and event-based systems [38] is to represent an event by a set of attribute value pairs whereas a primary attribute may be used to characterize the type of the event. For instance, an event of type weather may carry information related to attributes humidity and temperature.

More formally an event consists of an event type and a set of attribute value pairs, i. e., $e = \langle type : name \mid a_1 : v_1, \ldots, a_d, : v_d \rangle$, whereas for each specific attribute name a_i, v_i can accept values in domain D_i. Wherever the event type is not needed, we will also use the simpler notation for events $e = \langle a_1 : v_1, \ldots, a_d, : v_d \rangle$. The set of recognized attributes $\{a_1, \ldots, a_d\}$ of an event-based system and its values in $\mathbb{D} = \Pi\, D_i$ span the possible event space of an event-based system. Therefore, any event $e \in \mathbb{D}$ can be geometrically represented as a d-dimensional point in the event space $\mathbb{D}$.

The key objectives of event-based systems are to be informed, but also to understand the cause and correlation of events. This requires events to carry timestamps to decide on their order and the time of occurrence. A logical timestamp, e. g., a sequence number, allows ordering events while a chronological timestamp indicates the actual wall clock time when the event has occurred. Especially when detecting temporal correlation, we will build on the fact that events carry

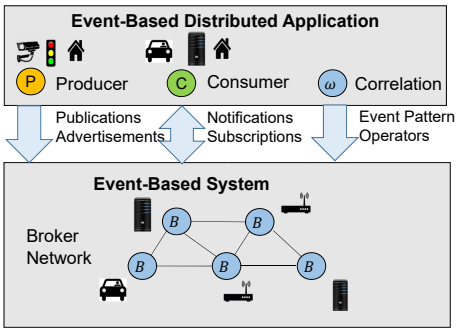


Figure 2.1: This figure illustrates the basic interactions between a distributed application and an event-based system.

timestamps to deterministically order and detect temporal correlation between events.

2.2 DISTRIBUTED SYSTEM MODEL

A distributed application consists of producers P_i and consumers C_j of events (Figure 2.1). Producers announce in the form of *advertisements* events they intend to publish. Whenever a producer observes an event, it will *publish* the event to the event-based system. Consumers register their interest in the form of a *subscription* to the event-based system and will be notified about published events of interest. The events can be primary events published by the producers or complex events which are detected by the event-based system. The distributed application can define how events are correlated, by defining *operators* that will be executed over the event streams.

A set of *brokers* $\{B_1, \ldots, B_k\}$ —forming together a *broker network*— accomplishes the functionality of routing events from producers to consumers and detecting correlations. Brokers can be any entity of the infrastructure capable of processing the logic of the operator. In this work, we consider a wide range of computational resources, including end-devices, servers, and network elements, e. g., programmable switches.

2.3 SUBSCRIPTIONS AND ADVERTISEMENTS

A *subscription* s posed by a consumer C defines which events are of interest to C. A consumer can express interest in an event by specifying a *subspace* $s \subset \mathbb{D}$ that constraints the attribute values of interest to the subscriber. Figure 2.2 illustrates the geometric relationship between subscriptions and events within the event space. While an event is a d-dimensional point within the event space, the subscription is a subspace in which events *matching* the subscription are comprised. For the time a subscription s is valid, C will receive an event stream

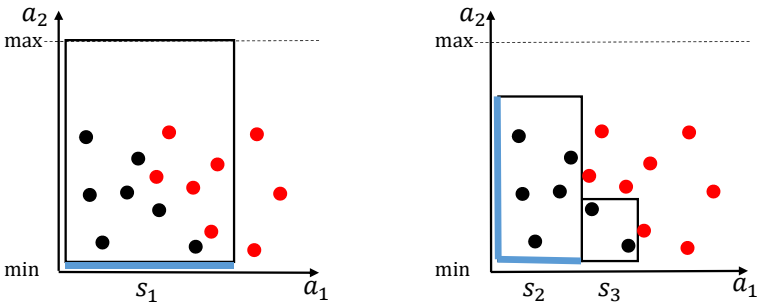
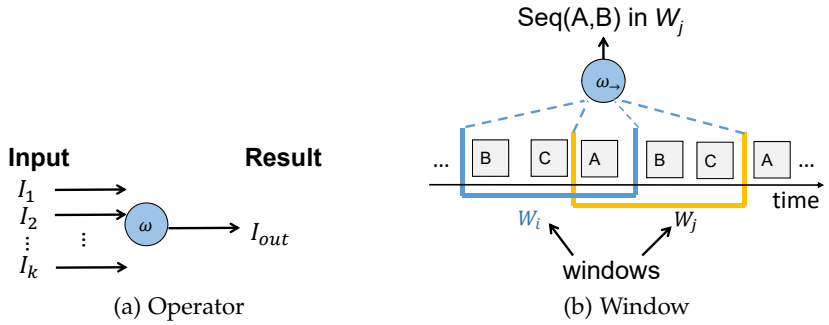


Figure 2.2: This figure illustrates the geometric relationship between event and subscription formed by an event space comprising two attributes.



(a) Operator (b) Window

Figure 2.3: This figure shows the basic principle in correlating events from incoming streams at an operator. The operator detects a pattern —here a sequence of event A followed by event B— restricted to a window.

comprising events matching s ordered by the logical timestamps. We will denote by S the set of subscriptions which are registered with the event-based system and refer by $|S|$ as the number of registered subscriptions of an event-based system.

Similar an *advertisement* adv is an announcement by a producer P to specify in which subspace of the event space, P intends to publish events. Therefore, any e published by P with adv should yield $e \in adv$. We will denote by A the set of advertisements registered with the event-based system and refer by $|A|$ to the number of registered advertisements. Advertisements and subscriptions can be used by the brokers of the event-based system to establish bandwidth efficient routes between producers and consumers of events. We measure the overhead dependent on $|A|$ and $|S|$.

2.4 OPERATOR MODEL

An *operator* ω produces an outgoing *event stream* I_{out} based on events received from its input streams $I_1, \ldots, I_k$ (cf. Figure 2.3a). The operator works on a partial substream of its incoming events named *win-*

Entity	Notation	Entity	Notation
producer	P	consumer	C
broker	B	operator	ω
set of operators	Ω	stream	I
operator graph	$G(\Omega, I)$	advertisement	adv
set of advertisements	A	subscription	s
set of subscriptions	S	(complex) event	e
attribute	a	value	v
domain	D	window	W
event space	$\mathbb{D}$		

Table 2.1: Notation used within the thesis

dow and aims to detect an *event pattern* comprised in the window (cf. Figure 2.3b. The operators execute a sequence of *correlation steps*. At the beginning of the correlation step, the operator updates its window W, i. e., the operator selects the events of relevance for the correlation step. For instance, in Figure 2.3b after updating to window W_i, the operator performs the next correlation step with respect to the events B, C, and A. The operator will check whether the window comprises a specific pattern, e. g., a sequence of A followed by a B. A correlation step ends when a pattern can or cannot be successfully detected. If the pattern can be detected in the window, then the operator will produce based on the pattern one or multiple complex events. In the example, $Seq(A, B)$ is not comprised in W_i, but after updating the window to W_j, the pattern can be detected. For updating and defining windows over an event stream, event processing systems offer many different policies, e. g., selecting a count of events or based on the timestamp. In addition, event processing systems provide policies that allow evicting events from windows (consumption policies), and resolve ambiguities in event patterns (parameter contexts). Event processing systems also offer timestamping mechanisms for the complex event, e. g., the timestamp for the complex event will be the largest timestamp of an event comprised in the detected event pattern.

The outgoing stream of an operator can serve as an input stream of another operator. Therefore, the multiple dependent operators can form an operator Graph $G(\Omega, I)$ where streams in I interconnect operators in Ω. Any broker of the event-based system can execute the operators in Ω.

In Table 2.1 we give an overview of the basic notation used in the remainder of the thesis.

3 EVENT DISTRIBUTION

Event distribution is the process of forwarding events from an event producer to all event consumers with a matching subscription. In particular, publish/subscribe is a key paradigm for establishing a decoupling between event producers and event consumers. In this chapter, we will motivate and illustrate important aspects that arise in the context of the event distribution problem and help publish/subscribe to meet Quality of Service (QoS) requirements under high dynamics (cf. research questions RQ1.1-1.3)

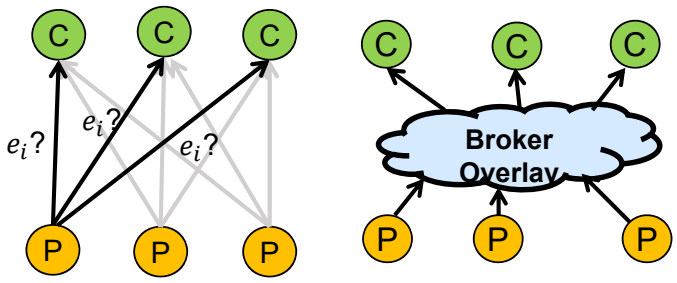
We focus on the design of broker overlays and corresponding routing mechanisms (cf. Section 3.1), motivate the need for adaptation between fine-grained and coarse-grained approximation of subscriptions to meet QoS requirements (cf. Section 3.1.2), and illustrate how underlay specific abstractions help to improve bandwidth-efficiency and latency (cf. Section 3.1.3). After an overview of the state of the art, we explain the following own contributions to meet QoS under high dynamics (F1) as well as to support high performant event distribution (F2):

F1: Muhammad Adnan Tariq, Boris Koldehofe, Gerald Georg Koch, Imran Khan, and Kurt Rothermel. "Meeting subscriber-defined QoS constraints in publish/subscribe systems." In: *Concurrency and Computation: Practice and Experience*. Wiley, 2011, 23.11, pp. 2140–2153. DOI: 10.1002/cpe.1751

F2: Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler, and Kurt Rothermel. "High Performance Publish/Subscribe Middleware in Software-Defined Networks." In: *IEEE/ACM Transactions on Networking*. IEEE, 2017, 25.3, pp. 1501–1516. DOI: 10.1109/tnet.2016.2632970

3.1 THE EVENT DISTRIBUTION PROBLEM

In publish/subscribe a decoupling of event producers and consumers is achieved by relying on a network of interconnected brokers. The brokers collaboratively manage an event distribution structure composed of directed transport-layer links between pairs of brokers, this way forming a *broker overlay*. The formation of a broker overlay is accomplished by the brokers collecting from producers and consumers information on subscriptions and advertisements. As a consequence, producers and consumers do not need to keep information on each other in order to publish and receive events (cf. situation in Fig-



(a) Without brokers (b) Broker network

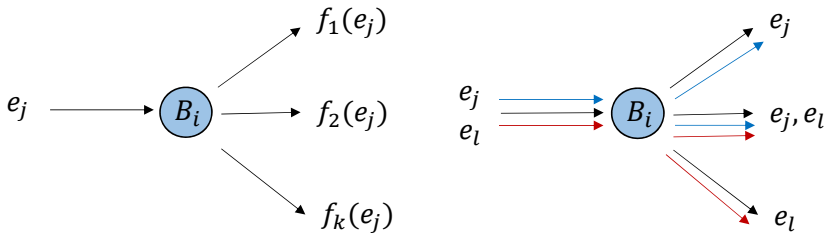
Figure 3.1: Without a broker network each producer needs to keep track of its consumers and ensure events are forwarded

ure 3.1a). The work in forwarding events is delegated and shared by the broker network (cf. situation in Figure 3.1b).

The mechanisms for managing and adapting the broker network need to ensure correctness and fulfill the QoS requirements of producers and consumers. Most importantly, (R1) the broker network has to ensure that every event published by a producer is received by any consumer with a matching subscription. Furthermore, important QoS requirements like (R2) end-to-end latency and (R3) bandwidth-efficiency need to be ensured by the mechanisms of a broker network. While publish/subscribe mechanisms comprise many more correctness criteria and QoS attributes, we focus here for the sake of understandability on requirements R1–R3. Nevertheless, further aspects like resilience to failures, mobility, and security are discussed in the context of subsequent chapters in the context of event processing.

The key problem for event distribution is to ensure correctness and QoS (i. e., R1–R3) under dynamics that occur during the execution of a publish/subscribe system. The causes of dynamics are manifold. New subscriptions or unsubscriptions can occur while event distribution takes place. Fluctuations in the availability of network and processing resources can require changes in the broker overlay. The occurrence and therefore the rate of produced events can highly fluctuate. The many faces of dynamics can occur in combination. This makes it extremely challenging to meet correctness and QoS and ensure the overlay remains scalable with many consumers and producers.

In the following we introduce three key aspects: i) *routing and overlay structures* that decide how events are routed through the network, ii) *the subscription granularity* that influences the overhead and efficiency at which an overlay can forward events, iii) *underlay awareness* in order to improve bandwidth-efficiency and accelerate the distribution of events.



(a) Filtering (b) Channelization

Figure 3.2: Filtering and channelization

3.1.1 *Event routing and Broker Overlays*

Event routing is a mechanism performed by each broker to decide on which of its outgoing links (determined by the broker overlay) event messages are forwarded. An event routing mechanism in combination with a broker overlay must ensure that all consumers receive those events matching their subscriptions. More precisely, the overlay management has to ensure that there exists at least one path from every producer to every consumer in the overlay. In addition, the routing algorithm has to ensure for every possible produced event that the corresponding distribution tree—rooted at the producer of the event— covers all consumers interested in the event. Recall, each event can be of relevance to a distinct set of consumers and therefore uses different paths to reach all its consumers.

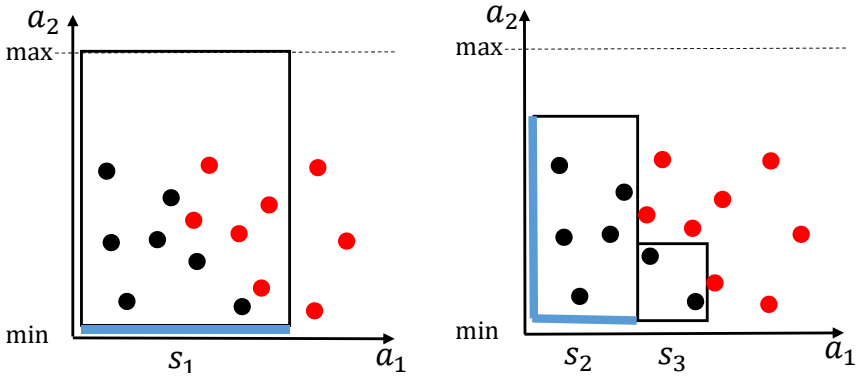
Concerning QoS, the path length and the quality of communication links are important metrics for an overlay. Both metrics influence bandwidth usage and end-to-end latency. Similarly, event routing itself impacts the ability of a publish/subscribe system to meet QoS properties. For example, to be highly bandwidth-efficient in the course of event routing, the broker may install on each of its outgoing links a filter operator (cf. Figure 3.2a). As an alternative, the overlay may color its outgoing links by classifying events by types and corresponding channels (cf. Figure 3.2b), which reduces the time in performing forwarding decisions.

In the presence of dynamics, it is therefore important that event routing and overlay mechanisms

1. ensure the connectivity of the overlay even under frequent changes,

2. reduce the number of unnecessary messages (transmissions) for bandwidth-efficiency,

3. ensure the processing overhead and the path length corresponds to the latency and bandwidth constraints specified by consumers and brokers.



(a) Coarse-grained subscription using one subscription on one dimension

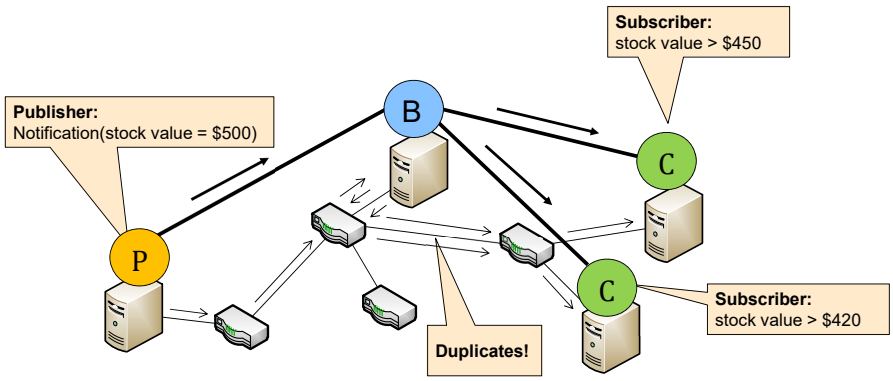
(b) Fine-grained subscription using two subscriptions on two dimensions

Figure 3.3: The figure shows different ways to define subscriptions for events of interest to a consumer (black) by using coarse and fine-grained subscriptions.

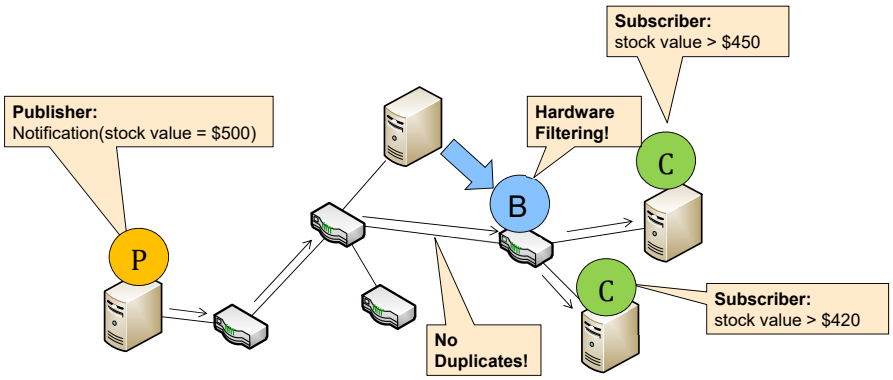
3.1.2 *Subscription Granularity*

To define the interest in events there exists both *coarse-* and *fine-grained* subscription models. A coarse-grained subscription limits the number of attributes of the event space to describe the interest in events. Furthermore, a coarse-grained subscription model can limit the set of possible restrictions on attribute values, e. g., by using categories or classes as known for topic-based publish/subscribe. The advantage of applying coarse-grained subscription models is a simpler and faster encoding as well as performance gains for matching events against subscriptions. For instance, in Figure 3.3a the coarse-grained subscription comprises only a single attribute. At the downside a coarse-grained subscription may lead to additional overhead in terms of bandwidth usage, e. g., in Figure 3.3a the events in red will be forwarded to the consumer with subscription s_1 although they are not useful to the consumer. Contrary, a fine-grained subscription can help to reduce unnecessary transmissions of events (we refer to as *false positives*) but leads to additional complexity in filtering events and processing and encoding subscriptions. For example, Figure 3.3b requires to match every event against two dimensions using multiple subscriptions to avoid unnecessary transmissions.

Whether a more coarse- or fine-grained subscription is suitable depends not only on the concrete set of subscriptions but on the concrete event load generated by the producers and also at the concrete resources and their usage. Therefore, under changing conditions of the event load or the usage of resources, adapting the subscription granularity can be an important approach to meet the QoS requirements of consumers. However, adapting the granularity of subscriptions also requires to



(a) Overlay-based broker network



(b) Underlay-aware broker network

Figure 3.4: The figure shows potential efficiency gains when event routing becomes underlay aware.

1. minimize the impact of adapting subscription granularity on reconfiguring broker overlay,

2. represent subscriptions such that the coarsening and refinement of subscriptions to meet the QoS requirements are facilitated.

3.1.3 *Underlay awareness and in-network operations*

In order to provide QoS for an overlay network, it is essential to understand the properties of overlay links including their capacities and delays. This is a very complex task since the performance of a link depends on a number of distinct mechanisms which are not visible above the network transport layer and can even change. In addition, the sharing of resources and therefore the occurrence of capacity limits cannot be controlled by the broker overlay. However, a rough estimate of the performance of an overlay link can be obtained by monitoring its conditions. To cope with uncertain state information, the overlay may introduce sufficient redundancy to overcome bottlenecks at specific links and ensure that the performance gracefully degrades with the dynamics of the overlay.

In modern network infrastructures, Software-defined Networking (SDN) offers abstractions to configure the underlying network and control the network traffic. In addition, Software-defined Networking (SDN) allows bringing functionality on high performant hardware components that are commonly deployed on network elements like switches, e. g., performing line-rate packet header processing. Therefore, SDN networks have also a high potential to improve the performance of broker overlays by avoiding redundant messaging and by accelerating the processing speed for events by offloading broker functionality to the network.

Figure 3.4 illustrates possible efficiency gains that can be accomplished by a Software-defined Network. In Figure 3.4a the processing and forwarding of events are performed without awareness of the underlying network link. In consequence, messages are sent multiple times over an underlay link, and in every processing step the event needs to be processed by logic which resides at the application layer. In Figure 3.4b the broker functionality is realized by a network element, i. e., a switch, avoiding duplicates of event messages on communication links, but also accelerating the speed at which events are processed. This requires

1. appropriate representations of events and subscriptions in order to be able to perform event matching at the network layer,

2. methods for efficiently controlling and adapting the data plane dependent on dynamic subscriptions and unsubscriptions.

3.2 RELATED WORK

The decoupling of producers and consumers —and this way building scalable distributed applications— has been a key motivation for publish/subscribe. For building suitable and scalable broker networks, bandwidth-efficiency by utilizing the diversity of producers and consumer has become one of the dominating research directions in building publish/subscribe for many producers and consumers.

Dependent on the application domains, many different paradigms like topic-based publish/subscribe and content-based publish/subscribe [38] have evolved. Each of the paradigms supports a different level at which diversity of information can be utilized, which strongly relates to the expressiveness of the corresponding subscription models. The idea of topic-based publish/subscribe [92] is to classify events by distinct categories, e. g., a twitter hashtag that categorizes the channel over which corresponding events can be received. Realizing such a channel typical corresponds to forming a group of brokers, e. g., building a multicast group to efficiently forward events with respect to the channel. Extensions of the model have also pro-

posed hierarchies [8] which can be useful in order to describe events with respect to organizations or locations.

However, for many applications where the interest in events depends on the attribute values, the content-based publish/subscribe model —we already introduced in Chapter 2— offers more flexibility by restricting the interest to ranges of values along multiple dimensions. Content-based publish/subscribe systems like Gryphon [11], JEDI [29], Hermes [87], Siena [24], Rebeca [74], Padres [52], Spine [20] and SpoVNet's Event Service [108, 112] demonstrated that the content-based subscription model is the key to highly bandwidth-efficient communications.

There are two key principles for accomplishing content-based publish/subscribe, i. e., to build on in-network filtering [24, 107] and clustering of similar subscriptions [106]. In addition, the design of broker networks has been driven by being highly robust to dynamic changes in the subscriptions. Accomplishing a decentralized organization of the broker overlay, as accomplished in our contribution [107], allows for building a highly scalable broker overlay which is entirely self-managed by the consumers and producers. In addition, to counteract dynamic situations recent work has also looked into introducing more flexibility in subscription models by allowing for parameterization [55], i. e., dynamically changing the consumers interest dependent on the context at hand and even to transit between multiple subscription models, i. e., topic-based and content-based at run-time [90].

Since event-based systems are often coupled with physical systems and business process, e. g., financial transactions, supporting further QoS attributes such as end-to-end latency [108], reliability [5] and security (cf. Chapter 7) has been an important research direction. In reducing end-to-end latency, a key bottleneck for the performance is that the broker network is realized as an overlay without knowledge on the physical communication infrastructure. Therefore, traditional network mechanisms like IP-Multicast can outperform publish/subscribe, however, at the cost of being not bandwidth-efficient. An important direction of work has been to monitor the underlying network structure and make the broker overlay agnostic to the underlying communication model [32, 56, 63, 109]. Further improvements, can be accomplished by programming network hardware devices [57, 93]. In our contribution [17] we build on abstractions for Software-defined Networking in order to manipulate the routing tables inside the switches and accomplish high performance for content routing. Building on this findings recent works have looked further how to efficiently control software-defined publish/subscribe systems in a scalable way [18] and benefit also from a mix of a programmable network and a traditional broker infrastructure [16]. Further innovations in SDN, like the popular P4 language, offers additional abstractions to program the pipeline of network switches [59].

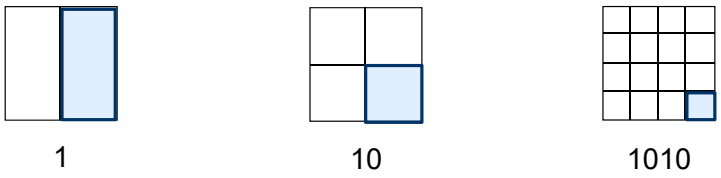


Figure 3.5: Representation of dz-expressions to address coarse- and fine-grained subspaces

3.3 CONTRIBUTIONS

In this section, we will summarize two contributions of this thesis that help to improve the ability of broker overlays to adapt under dynamics to support latency and bandwidth-efficiency of publish/subscribe systems.

The first contribution [107] presents an approach to building a highly decentralized broker network, that can dynamically adapt the subscription granularity. This way consumers can fulfill latency constraints by adapting their bandwidth usage. The second contribution [17] presents an approach that allows offloading the broker functionality to the Ternary Content Addressable Memory (TCAM) of switches in SDN. As a result events are processed at line-rate and end-to-end latency is significantly reduced. Both contributions share a common subscription model based on spatial indexing.

3.3.1 *Subscription Model*

The spatial indexing model approximates a subscription by a set of binary strings of variable length, say $DZ = \{dz_1, \ldots, dz_n\}$, where each string dz_i corresponds to a subspace of the event space spanned by its attributes. In general, the relationship between dz_i and dz_j can be described as follows: if dz_i is a prefix of dz_j, then all events of the subspace spanned by dz_j are comprised in the subspace spanned by dz_i. In particular, the empty string ϵ represents the entire event space. The string can be generated by a step-wise partitioning of the event space along each attribute dimension where ″0″ corresponds to a left/lower half of a partition and ″1″ to a right/upper half (e. g., the string ″1″ corresponds in Figure 3.5 to the right partition).

Consequently refining a subscription is accomplished by increasing the length of dz in DZ while coarsening a subscription can be accomplished by reducing the length of subscriptions (e. g., in Figure 3.5 the subspaces ″10″ and ″1010″ are both refinements of ″1″). It is important to note that two dzs of the same length are non overlapping. Furthermore, two dz_1 and dz_2 are overlapping iff either the subspace of dz_1 is completely contained in the subspace of dz_2 or vice

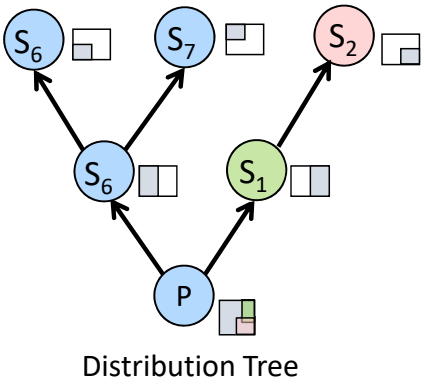


Figure 3.6: A subtree connecting consumers *dz*-strings covered by a producer *P*. The coarseness of a dz-determines the hop distance to the producer

versa. Therefore, the subspace of an empty string ϵ supersedes all subspaces of the event space.

3.3.2 *Adaptive Overlay meeting latency and bandwidth constraints*

In [107], we build exactly on the relationship between *dz*-strings in the subscription model in order to establish an overlay network. The brokers in the network are the consumers themselves that perform event routing with respect to each subspace $dz \in DZ$ defining their subscriptions. For each dz a consumer maintains parents with a superseding subscription and children with refined *dz*-strings. This ensures that once the parent of a consumer with dz receives all events for its covering subscription, the consumer of dz will also receive all events of interest. Similarly, a producer connects for each of its *dz*-strings approximating its advertisement the root of the subtree comprising all dz' superseded by the subspace of dz (cf. Figure 3.6).

The granularity of the subscriptions controls the hop distance between a consumer and a producer. Coarse-grained subscriptions of a consumer are close to the producers, i. e., close to the root of the distribution tree. Fine-grained subscriptions are close to the bottom of the overlay network and impose therefore a longer distance to the root of the distribution tree. Since the subspaces are either contained in each other or completely disjoint, every consumer only forwards events that correspond to the interest of the consumer. In the article [107], we show that the overlay management helps in fulfilling individual latency requirements of consumers. Moreover, we explain that the described mechanisms are very robust to high dynamics in changes to the subscriptions.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

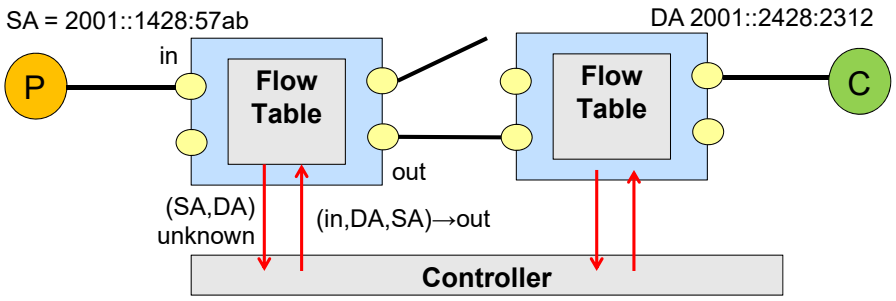


Figure 3.7: The controller installs forwarding rules in the FlowTables of the Switch. The rules ensure that every advertised event is reachable to all its consumers. For IP_{fix} no rules are installed and therefore all subscriptions and advertisements are forwarded to the controller.

3.3.3 *Underlay awareness with software-defined networking*

In [17], we propose a publish/subscribe system named *PLEROMA*. *PLEROMA* accelerates brokers by executing the broker functionality directly on a network switch which forwards events at line-rate through the network. Offloading the broker functionality to a network switch does not require any modification of the functionality of the switch itself. We build solely on the control abstractions offered by the OpenFlow standard. Note, many network devices currently support the OpenFlow standard by the Open Network Foundation (ONF).

Again we use the *dz*-string encoding to represent and approximate the events, subscriptions, and advertisements. The *dz*-string is embedded in a network packet by replacing a specific header field of the packet. For instance, the IPv6 multicast range offers sufficient space to embed a sufficient number of *dz*-strings. Since a publish/subscribe middleware can be seen as a replacement for multicast services and does not interfere in the address space of other protocols, it can hence coexist with other network applications, e. g., unicast traffic. However, in general, any packet header supported by the OpenFlow standard can be used to encode an event comprised in a network packet.

For every event published, the controller has to ensure that there exist corresponding forwarding rules at the Flow Tables of all switches connecting the producer and all interested consumers (cf. Figure 3.7). Such a rule comprises a prefix of the event, i. e., a *dz* of shorter length. Network switches typically possess a highly efficient hardware implementation named TCAM that supports the prefix matching in a single

clock cycle. Hence, events routed through the network are processed in line-rate without requiring any additional processing at a broker.

In order to establish corresponding forwarding rules, the controller needs to be informed about subscriptions, unsubscriptions, and advertisements. For this purpose, a specific IP-address, named IP_{fix} is reserved, which consumers and producers use in the packet header in order to indicate dynamic subscriptions, unsubscriptions, and advertisements. Once such a packet is received by a switch it is forwarded to the controller which updates the event distribution structure (similar to the broker overlay of Section 3.3.2) and updates the flows of the switches. In doing so, *PLEROMA* contributes with concepts to distribute the controller logic, to support the cooperation of multiple domains, and to minimize the utilization of the TCAM —a scarce and expensive resource. In experiments, we showed for virtualized switches as well as for hardware switches, that offloading brokers to the network give tremendous gains with respect to end-to-end latency. While traditional overlay networks typically operate in the order of milliseconds, with PLEROMA's implementation we could measure end-to-end latency in the order of microseconds.

4

OPERATOR EXECUTION

The performance of an event processing system strongly depends on where and how the deployed operators are executed. In this chapter, we will focus on the problem of ensuring predictable performance of event processing under varying event load. To this end, we address the problem of adaptive operator execution. We focus on how to adapt the rate at which operators can consume and produce events by studying methods for the parallel execution of operators. This chapter builds on the following contribution:

F4: Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. "Predictable Low-Latency Event Detection With Parallel Complex Event Processing." In: *IEEE Internet of Things Journal*. IEEE, 2015, 2.4, pp. 274–286. DOI: 10.1109/jiot.2015.2397316

4.1 THE ADAPTIVE OPERATOR EXECUTION PROBLEM

An event processing system executes the processing logic required to detect events. The processing logic determines the processing steps needed to derive outgoing streams of complex events. The processing logic —often specified by a SQL-like query— can be decomposed into several dependent operators forming an operator graph $G(\Omega, I)$ comprised of operators $\omega \in \Omega$ and streams $I \subset \Omega \times \Omega$ interconnecting the operators. For instance, Figure 4.1 illustrates an operator graph detecting changes in the status of road segments (congested / non-congested) by i) aggregating measurements on the speed of vehicles on a specific road segment over a time-restricted window, ii) detecting changes in the aggregated values that correspond to a changed status of a road segment, iii) and filtering status updates of road segments of interest to consumers for updating their individual traffic routes. The event processing system has to allocate resources for the execution of each operator. In the context of Figure 4.1 this can be typical IoT resources, like things, edge servers or cloud servers. The process of mapping the operators to processing resources is called *operator placement*. The placement influences the ability of event processing to meet QoS in multiple ways. For instance, the computational speed of processing resources and their utilization determines the rate at which events can be processed and therefore the achievable throughput in processing events. In addition, the location and communication interfaces to interconnect the operators determine communication latency and bandwidth usage.

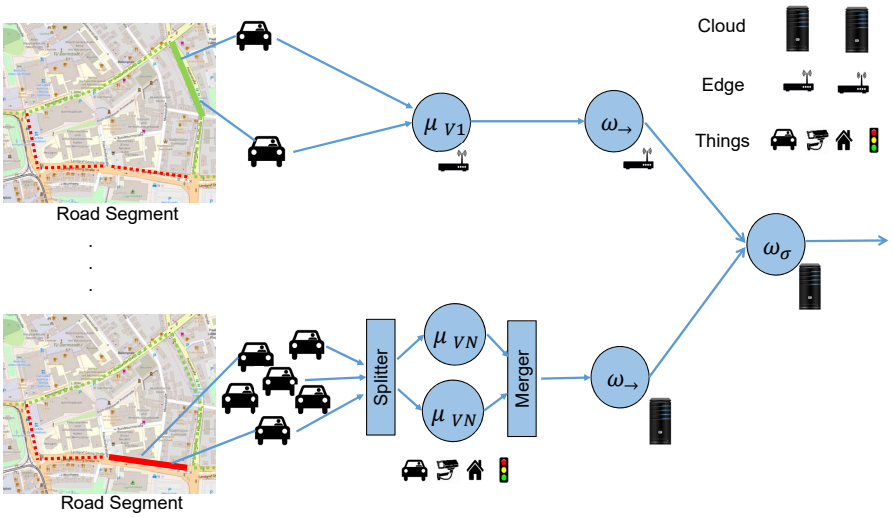
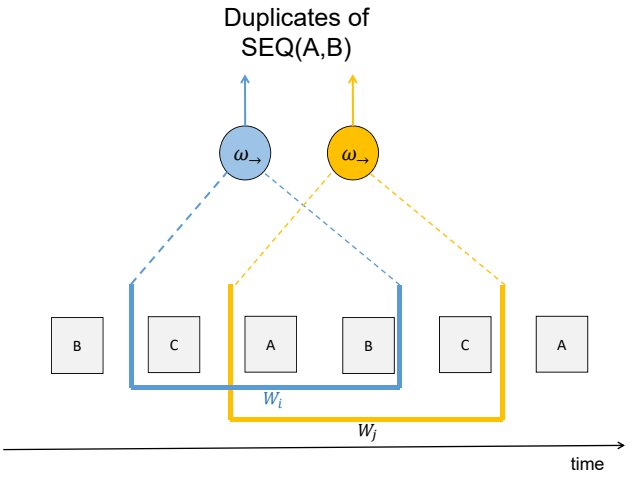


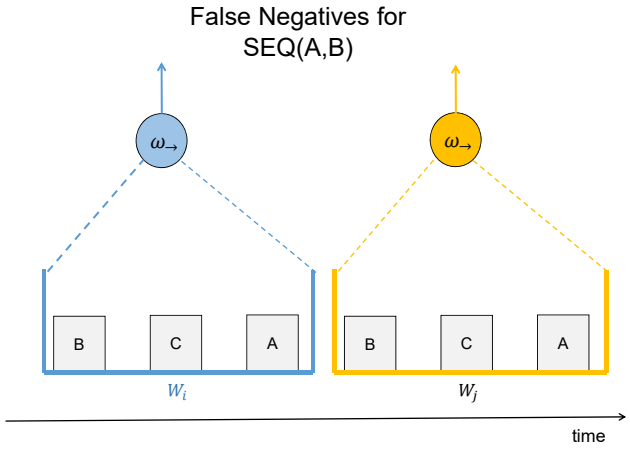
Figure 4.1: An operator graph for detecting changes in the status (congested, non-congested) of road segments. The operators are placed on IoT resources at distinct locations. Some operators can be parallelized in order to increase the throughput in processing events.

In this chapter, we focus on offering predictable end-to-end latency for the execution of operators under varying event load. To understand the challenge, consider the situation in which the rate at which events arrive at an operator overloads the computational resource hosting the operator. The queuing time of events will increase and the latency until the corresponding complex event can be detected will exhibit increased delays. In order to mitigate the effects of overload situations, the event-processing system needs a way to scale the computational resource to adjust its processing rate to the arrival rate of events.

One way to achieve this is to migrate the operator to a different computational resource which increases the processing rate of the operator, i. e., improve the operator placement with respect to throughput. However, scaling physical computational resources is limited to the maximum processing capabilities of the available resources. For example, in Figure 4.1 the best processing rate is bounded by the best available server instance, most likely located in the cloud. Furthermore, moving an operator to a remote location may decrease the efficiency of an operator placement with respect to bandwidth usage, e. g., transmitting all measurements to a cloud server. An alternative is to increase the number of computational instances hosting the operator by means of parallelization. For instance, the operator μ_{VN} is realized by multiple operator instances concurrently operating on the same incoming event stream. This raises two problems, namely how to best parallelize the operator logic (cf. Section 4.1.1) and how



(a) Duplicates



(b) False negatives

Figure 4.2: The figure illustrates how splitting policies impact the correct detection of complex events.

to adapt the parallelization degree (cf. Section 4.1.2) in order to meet end-to-end latency bounds.

4.1.1 *Operator Parallelization*

In order to adapt to varying load, the parallel execution of an operator provides means to scale the processing resources of an operator. This poses the problem of introducing a dynamic level of concurrency which can be adjusted to the changes of the event load.

A most basic parallelization technique —similar to writing concurrent programs— is to parallelize the steps performed by the event processing logic. The achievable parallelism depends on the concrete operator. To understand some inherent limitations of this parallelization technique in more detail, consider the sequence of windows of events $W_1, \ldots, W_n$ processed by an operator ω. Following the event

processing model introduced in Chapter 2, each window W_i possibly comprises one event pattern to be detected in a specific processing step of ω. Furthermore, let the corresponding processing logic f_ω define the mapping of each W_i to the set of produced events after the completion of a processing step. For each window W_i, $f_\omega(W_i)$ can be parallelized by several threads working cooperatively in order to produce events. For instance, a parallelization of f_ω could be obtained by describing the detection logic as a finite state automaton (FSA) [98]. A partition of the distinct states of the automaton exhibits a pipeline parallelism that allows to speed up the execution of f_ω utilizing multiple threads. Note, that the parallel execution of $f_\omega(W_i)$ will reduce the time to process a specific window. However, the speedup is unrelated to the arrival rate of windows that need to be processes, therefore a significant increase in event load may not be compensated by changing the parallelization degree. For instance, in the parallel FSA model the parallelization degree is bounded by the number of states of the FSA model.

A promising alternative is to assign to each window W_i —once it is available at ω, a distinct operator instance with the same processing logic but own processing resources. This would bound — always assuming a sufficient number of parallel working operator instances is available— the end-to-end latency by the worst-case execution time for $f_\omega(W_i)$. A challenge for this technique, named *data parallelism*, is to deal with the dependencies of subsequent windows and consequently find appropriate partitions of the event stream to substreams processed by the operator instances. In particular, the partition model has to determine split points in the event stream that ensure the substreams processed by an operator instance comprise all necessary events to detect the events of interest. Figure 4.2 illustrates exemplary problems in selecting a split point. The split points in Figure 4.2a lead to duplicates in detecting the same sequence of events A and B. In Figure 4.2b A and B are located in distinct partitions leading to false negative detections of events.

Key requirements for introducing operator parallelisms are therefore to find *partitioning models* that

1. can efficiently partition the event stream between a dynamic number of operator instances,

2. ensure that the events produced from sequential and parallel executions remain indistinguishable,

3. preserve the expressiveness of complex event processing in detecting event patterns.

4.1.2 *Adaptation of the parallization degree*

In addition to efficiently partition the event stream, the parallelization degree needs to be adapted dependent on the rate at which events arrive at an operator. While the central goal is to ensure predictable end-to-end latency, it is important to minimize also the resource usage. The resource usage is determined by the number of parallel operator instances. Therefore, we aim to avoid i) operator instances consuming unnecessary processing resources, and ii) overload situations due to under provisioning, i. e., by deploying an insufficient number of operator instances.

Under the premises that splitting and merging costs of streams remain negligible (e. g., can be performed at line-rate), the end-to-end latency can be modeled by a queuing system. As part of the queuing system, windows of events, i. e., W_i, will be enqueued in a waiting buffer until they are served by an operating instance. Each window has a service time which depends on the processing task, i. e., the concrete correlation logic, and the processing capabilities of the resource hosting the operator instance.

With queuing theory, one can model for a given probability distribution of events and an estimation of the service time, the probability that the queuing delay will remain below a latency threshold Δ_{max}. Consequently, to guarantee a latency of Δ_{max} with a probability p, a parallelization algorithm aims to choose the smallest number of instances that result in a probability $p' > p$ to satisfy Δ_{max}.

It is important to note that in general neither the event distribution, nor the characteristics of the execution environment, are known beforehand and depend very much on the application and computational resources available. In order to be able to provide predictable latency this requires

1. to correctly capture and predict dynamically the probability distribution of the event arrival time

2. capture performance profiles of operators and determine the service time for windows depending on concrete processing resources.

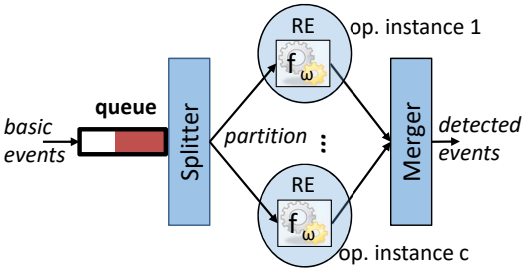
4.2 RELATED WORK

The need for the distributed execution to meet Quality of Service Constraints has been motivated by a significant number of research works in the context of the operator placement problem. One of the key problems addressed by contributions to the operator placement problem is to fulfill QoS demands while minimizing the cost in terms of resources [65]. A wide range of mechanisms for optimizing the operator placement have been proposed addressing different QoS de-

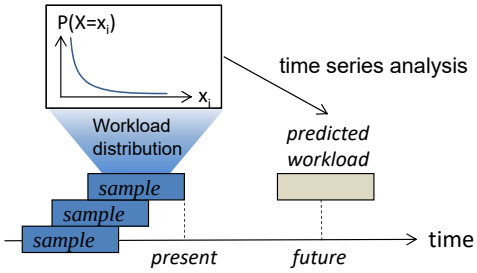
mands such as to achieve low latency [1], to minimize bandwidth [86, 91, 94], to lower message overhead [104], as well as to preserve trust and privacy [35] and even cope with dynamically changing QoS constraints [70].

To cope with performance bottlenecks due to dynamic event loads research works have proposed the parallelization of operators. For approaches addressing the parallelization of CEP operators one can distinguish between intra-operator parallelization [10, 45, 115] and data parallelization [47]. Both approaches deal with the difficulty that CEP operators build up internal processing state. Stateful operators prohibit applying simplistic parallelization algorithms [97] if inconsistencies are to be avoided. The principle behind intra-operator parallelization is to speed up the service rate of an operator by building a pipeline of dependent, but concurrently executable tasks implementing the operator logic. Since an intra-operator parallelization is highly operator dependent, initial works [45, 115] focused on how to find good parallelization for specific operators, e.g., join, sequence and aggregation operators. To reduce the effort in finding tasks to be pipelined, recent work [10] allows for automatically deducting them from a query specification building on a finite state machine (FSM) model for detecting events. Although intra-operator parallelization helps increasing the service rate, the degree of achievable parallelism is inherently bounded. In the underlying FSM model the maximum degree is bounded by the number of states in the FSM determined by the number of variables in the query.

Data parallelization [47] is a promising alternative to overcome the limitation posed by intra-operator parallelization regarding the degree of parallelization. The critical issue of data parallelization is to efficiently determine a suitable partitioning model. Almost all existing approaches that build on data parallelization, e.g., Nephele/-PACTs [13], Storm [6], Dryad [51], MapReduce Online [28], Stream MapReduce [22] and Schneider et. al. [98] establish key-based partitioning models. The main idea is to group events by means of a key, e.g., by a globally known location attribute. The splitter can partition the stream by deploying a filter for each of its parallel operator instances. The events of the incoming stream are then matched against the filters of the operator instances. However, this assignment is static, i.e., no dynamic changes to the assignment of key values to operator instances can be performed without the risk of losing operator state and therefore cause inconsistencies in the set of detected events. As a consequence, adapting the parallelization degree is hard to achieve for stateful operators. Furthermore, the capability for parallelizing the operators depends on the number of distinct keys that allow for a meaningful partitioning of the event stream. For many operators, a key that enables the grouping of events that belong to a pattern of interest is not available at all. This applies for instance to operators that



(a) Split merge architecture



(b) Prediction of work load distribution

Figure 4.3: This figure illustrates the basic approach to parallelize the execution of the event processing system.

match the occurrence of an aperiodic event bounded by two arbitrary events, as it is defined in the aperiodic operator in the Snoop event specification language [25]. Building on the contribution of this thesis, subsequent work has looked at optimizing the bandwidth usage by batching several windows [72]. In this way the overlap between different partitions and the overlap between different windows can be reduced.

4.3 CONTRIBUTIONS

In this section, we briefly summarize work of our own [71] that contributes to the dynamic scaling of operators execution to meet quality of service constraints. The work proposes a model for the efficient and scalable partitioning of event streams which ensures that sequential and parallel executions remain indistinguishable. Furthermore, building on the model the work proposes a method to dynamically adapt the parallelization degree under dynamic workloads. More specifically, the proposed approach ensures under dynamic inter-arrival times of events and a given QoS specification (Δ_{max}, p) the following: For all windows W_i arriving at operator ω, the period of time defined by the arrival of a window W_i and the detection of all event patterns comprised in W_i, is bounded by the delay Δ_{max} with probability p.

The parallelization approach follows the split-merge architecture depicted in Figure 4.3a. The operator maintains a set of dynamically

allocated operator instances, each operating on a specific partition of the event stream. The partitions and their assignment to operator instances is determined by the splitter. In order to support the efficient partitioning of event streams and ensure the parallel event stream remains indistinguishable from a sequential one, the partition model allows the specification of the splitting logic in form of two predicates that are applied to each arriving event. The predicates — if evaluated to true— open a new partition to be processed by an operator instance or close an open partition.

In case of the example, in Section 4.1.1 (cf. Figure 4.2) we can define the predicates for opening and closing a partition as follows:

1. The predicate $P_{open}(e)$ evaluates to true and opens a new window whenever $e = A$.

2. The predicate $P_{close}(e)$ evaluates to true if the maximum window size of a partition has been reached.

Since for each occurrence of A a new partition is opened, the execution will ensure that two instances of A are never comprised in the same partition. Therefore the situation of Figure 4.2a leading to duplicates is avoided. Moreover, any B that follows an A and complies with the window restrictions will also be comprised in the same partition, hence avoiding the problem of false negatives in Figure 4.2b.

Dependent on the context at hand, also more complex dependencies between subsequent windows can be modeled as known from expressive event correlation languages like Snoop [25]. For example, unlike with key-based predicates it is also possible to reason on conditions which depend on the context of an event, e. g., describing the window boundaries dependent on the occurrences of events. Most corresponding predicates can be realized by constant time operations and therefore constitute only little overhead in finding corresponding partitions.

In order to adapt the parallelization degree the work builds on a queuing system to determine the number of parallel instances dependent on p and Δ_{max} (cf. Figure 4.3a). The queuing system allows to determine a stable parallelization degree for a specific probability distribution of the inter-arrival time. Nevertheless, in event-based systems, the distribution of inter-arrival-times between events can change and therefore the parallelization degree must be adapted to meet Δ_{max} with a given p. To account for such changes the parallelization approach performs periodically a sampling of the workload distribution (cf. Figure 4.3b). By comparing the observed workload to different distributions, the best fitting distribution offering a worst-case approximation of the sampled work load is selected. The expected future distribution is predicted based on a time series prediction. Therefore, the correct number of operator instances can be deployed ahead of a window's arrival time. Similarly, following a profiling approach

the service time distribution for processing a window with respect to distinct processing resources is sampled.

The evaluation of the parallelization approach shows that depending on the overlap of windows, the splitter is capable of processing up to 100.000 events per seconds on standard server hardware. This determines the maximum arrival rate at which queuing delays at the splitter can be avoided. Furthermore, it is shown that the queuing model is capable of keeping event buffers below a threshold —in consequence ensures stable processing latencies— for a specified probability p. Contrary, reactive approaches accounting only for the current state of processing resources encounter very large variations in the buffer sizes leading at times to unpredictable delays in detecting events.

5 OPERATOR MIGRATION

In addition to dynamic changes in the event load (as discussed in Chapter 4), an event processing system has to adapt to dynamic conditions dependent on the availability of resources and the location of sources and sinks. If the available resources do not suffice to meet QoS requirements or new resources allow ensuring QoS at a lower cost, a change of the operator placement must be performed, and operators need to migrate to different resources. This poses the problem of migrating operator state in a non-disruptive manner between brokers of the broker network.

In this chapter, we focus on the problem of operator migration and context-aware adaptation of event processing systems. In particular, we consider mobility of producers and consumers as a key trigger for changing the operator placement. We highlight principles that allow event-based systems to operate in such a dynamic environment by introducing methods for performing mobility-driven reconfigurations of the operator graph and non-disruptive operator migrations. This chapter builds on the following contribution:

F5: Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. "MCEP: A Mobility-Aware Complex Event Processing System." In: *ACM Transactions on Internet Technology*. ACM Press, 2014, 14.1, pp. 1–24. DOI: 10.1145/2633688

5.1 THE OPERATOR MIGRATION PROBLEM

To illustrate the operator migration problem, recap the traffic information scenario motivated in Section 4.1 for which consumers are interested in road conditions in their proximity (cf. Figure 5.1). To define more accurately its interest, a consumer may define a context restriction in form of a range query, which is updated whenever the consumer's location changes. Therefore, the event processing system has to connect to new data sources dynamically. In order to meet QoS requirements with respect to latency and bandwidth usage, the state of the operator may need to be dynamically migrated dependent on the point at which producers and consumers access the network. As part of such a migration, a new target for the operator has to be selected, the state of the current operator needs be transferred and the migrated operator needs to connect to its incoming event streams.

A key objective when performing migrations is to ensure that migrations do not disrupt the execution of the event processing sys-

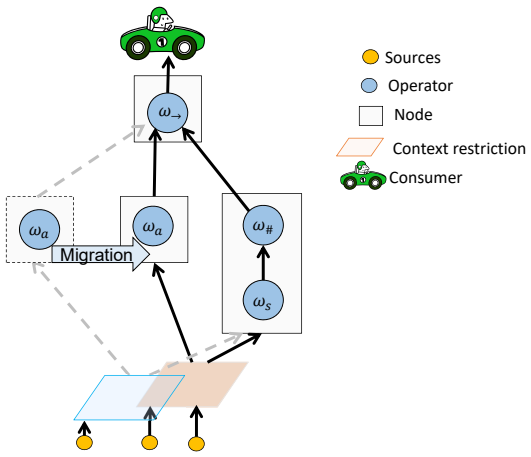


Figure 5.1: In this figure the interest in traffic information depends on the location of the consumer. With changes of the consumer's context, operators have to perform correlations with respect to a different set of event sources.

tem. Functionally, an execution with migrations should be indistinguishable from an execution without migrations. In consequence, the produced event streams must comprise the same events and preserve their order independent of the occurrence of migrations. That is particularly difficult in a mobile environment, where the processing steps depend on the consumer's context. Therefore, operators may frequently need to connect and disconnect to input streams that match the context of the consumer.

In this chapter, we therefore focus on two central problems. First, we will look into extending the query semantics so that the operators can take context changes of users into account, and dynamically select event streams dependent on the current context of users. Second, we deal with the problem of supporting migrations of operator graphs in such a dynamic environment.

5.1.1 *Query support for mobile producers and consumers*

In the standard operator graph model introduced in Chapter 2 producers and consumers are statically linked to the operator graph. In order to account for changes in the interest, e. g., for updating a context restriction on a region of interest, in a naïve solution the consumer unsubscribes and poses a new query. This imposes a number of complications —and in turn overhead— both for the consumers and the control of the event processing system. Consumers will encounter overhead because for location updates they need to initiate an unsubscription and subscription for every new query actively. The brokers of the event processing system will also encounter overhead. For every new subscription the event processing system has to de-

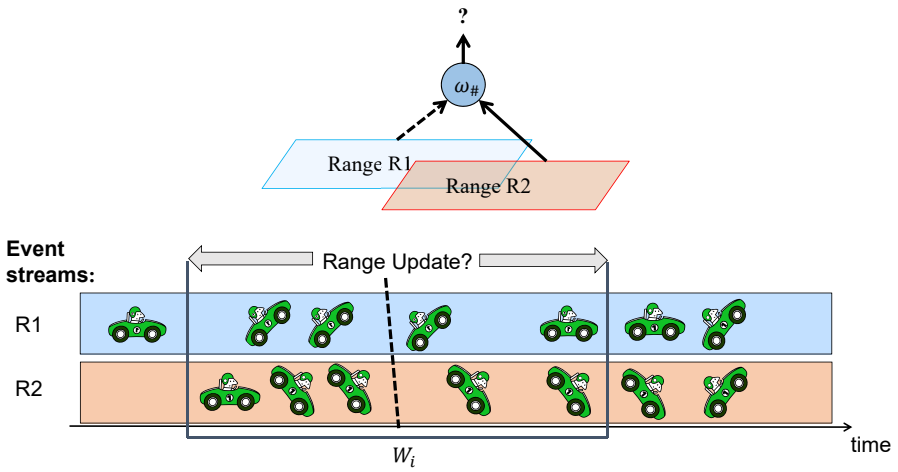


Figure 5.2: In this figure the state of $\omega_\sharp$ depends on the time when the range update is performed.

ploy the query. This costs overhead for the allocation of temporary redundant resources and imposes delays needed for initialization of the operators.

To counteract such effects, a worthwhile approach is to predeploy frequently used operator graphs. For instance, in the traffic scenario, each region of interest may operate its own instance of the operator graph. However, even for busy road segments, e. g., highways on the German road network, predeployment has been analyzed to be wasteful in the usage of resources [61]. The overhead measured in terms of necessary predeployments raises significantly with the precision at which the predeployment is required to cover the actual region of interest. For instance, consider a 2-dimensional spatial query. Reducing the error at which the query matches a spatial context restriction by a factor of 2, would increase the number of necessary predeployed operator graphs by a factor of 4 and for d dimensions by 2^d.

The alternative to supporting changing interests is to allow for a parameterization of the query by defining a context restriction which is updated automatically dependent on a dynamic context attribute of the user. Similar to the concept of information decoupling the application only requires to specify the context restrictions and thereafter receives all events of interest. The event processing system can allocate the resources which match the current demand of the consumers. In addition, it is also possible to reuse state, e. g., information that is available due to overlapping context restrictions.

In consequence, an event processing system with mobility support can improve precision and resource usage by a dynamic reconfiguration of the operators' state dependent on updates to the context restrictions. Whenever an update of the context restriction occurs, the event processing system needs to identify at an operator when to

switch the incoming event stream to match the new context restrictions and ensure, at the same time, the detected event stream does not contain false positives and false negatives, i. e., is consistent. The difficulties that arise to support consistent updates are illustrated in a simple example (cf. Figure 5.2). Given a context restriction in the form of a range query, the operator execution has to determine the point in time when to switch from the context-restricted event stream of R_1 to the context-restricted event stream of R_2. If this change occurs at an arbitrary point in time, W_i comprises events with respect to two different ranges which in turn can lead to false positive and false negative detection of situations (e. g., detecting or not detecting an obstacle on the left or right road lane).

Another question that arises in the context of the example is the definition of an appropriate consistency model. Consider that some events are detected with respect to R_1, others with respect to R_2. In what order should the events be delivered to the consumer? This requires

1. the suitable definition of a consistency model for the delivery of context-restricted data streams,

2. methods for reconfiguring dependent operators to comply with consistency models with dynamic context restrictions.

5.1.2 *Planning migrations*

Let us follow up on the example of mobile consumers with changing context restrictions. The consumer will change its access point to the network dynamically, in consequence, the available resources and the properties of resources can change, e. g., communication delay and bandwidth capacity to reach the resource. In response to such changes, an event processing system will dynamically update the operator placement to fulfill the consumer's QoS constraint and to improve the cost-efficient utilization of resources.

Finding a new placement imposes a delay. In particular, for many cost functions and constraints the optimal operator placement problem is known to be an NP-hard problem [58]. If the resource usage changes very dynamically —and in turn the optimal placement—, a better, but not optimal placement may allow anticipating changes faster. In addition to the time to plan for a better operator placement, additional time is needed to setup a new operator instance, migrate the current state to the new operator instance, and ensure that the event streams will be redirected.

The necessary amount of state to migrate an operator instance, can be significant, i. e., comprising internal state on detecting the event pattern, the windows of events ready to be processed by the operator, and often also contextual information that can be used to enrich

and help to detect event patterns, e. g., a database for recognizing faces on a camera stream. In order to avoid downtimes the event processing system will allocate additional resources for redundant operator instances and duplications of packets. Therefore, the migration costs are an important additional constraint to the operator placement problem. They also determine the time when an event processing system can benefit from an improved placement.

It is important to note that the amount of state to be transferred as part of a migration is varying over time. In particular, the internal state of the operator for detecting an event is typically empty after a window has been entirely processed by an operator. Therefore, the event processing system may restrict the points in time when a migration step is initialized or simply postpone a migration dependent on the progress in processing a window of events.

In order to ensure that necessary context information and streams are transferred in time to the new operator instance, the operator placement may also build on prediction models to anticipate future migration points, based on the updates to context restrictions. For example, considering the mobility pattern of the consumer, the event processing system can foresee the location updates of the consumers and predict the performance of a future placement.

This requires

1. methods for minimizing the amount of state to be transferred,
2. methods for planning migrations dependent on the mobility models of users.

5.2 RELATED WORK

Dynamic context restrictions have originally been explored in the context of *moving range queries* [44, 116]. A moving range query returns (sensor) data in a consumer-specified range and updates the obtained result upon changes to the location of a moving consumer. The context attribute that triggers updates is also denoted as *focal object*. For example, a consumer's moving range is always updated with the nearest taxis in a 500 m radius. These queries allow accessing filtered and aggregated sensor data [43, 105, 116]. However, the stateless semantics of a range query only allows operations that correspond to selecting events, but not to detect complex event patterns. Furthermore, a moving range query aims for finding all events of a range at a given time, but not all events that were produced within a temporal range. In order to detect event patterns reliably along the temporal dimension, it is important to ensure that all events selected in a window are complete and can be processed in temporal order.

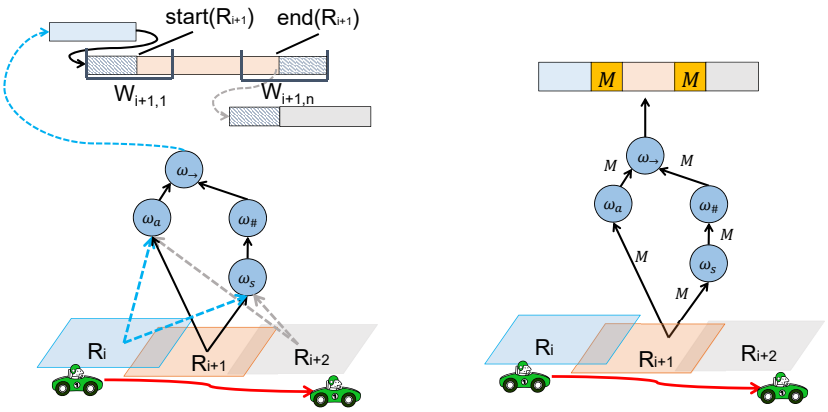
The issue to relieve the user in subscribing to a specific context restriction has also been addressed for parameterized publish/sub-

scribe systems [55]. Similar to moving range queries, the supported operations on event streams are in essence a selection of events, and therefore stateless operations. Realizing stateful operations over parametrizable publish/subscribe systems also requires consistently ordered event streams, respecting both the temporal order and the order of updates to context restrictions.

For the dynamic reconfiguration of operator graphs, approaches mainly focus on adapting the operator graph to dynamic event rates (cf. Section 4.1). In addition to the aforementioned parallelization of operators (cf. Chapter 4), solutions cover methods known from query optimization such as changing the order of operators [7] or the access pattern to event streams [33]. Per se, such mechanisms do not allow coping with the dynamics imposed by mobility-driven situational applications. Up to the proposal of Mobile Complex Event Processing (MCEP) [81] —a system of our own we detail in the next section—, the Borealis system [1, 37] offered the best support for mobile event processing. Borealis allows for on-the-fly query modifications by altering or replacing processing components, e. g., when an operator is moved to a new location. Furthermore, the system also supports time-travel methods which rewind a continuous query to access historic event streams, an important concept to initialize the state of an operator after a migration. Nevertheless, to comply with a consistent operation in the presence of changes to the context restriction, it is important that each operator can determine the state and required updates to reconfigure in an autonomous way. Furthermore, as part of operator reconfigurations there is a huge optimization potential by building on dependencies between consecutive updates of the context restrictions, e. g., the overlap in event streams between two ranges R_1 and R_2.

Improvements in end-to-end latency and communication costs are achieved in these systems by placing operators on computing nodes close to sources to minimize network traffic and delay [31, 48, 65]. Most of these approaches are tailored for scenarios with infrequent changes, i. e., where sensors and consumers are not mobile. Therefore, either the impact of migration costs on the placement strategy or amortization of these costs are not considered.

Recent works —in parts building on the contribution of this thesis— investigated the problem of dynamically migrating operators in a device-to-device communication model. In doing so, the approach by Starks et. al [104] optimizes the availability of event streams under the high dynamics of the network topology formed by the directly communicating devices. Dwarakanath et al. [36] deeply analyze the state model that allows for fine-grained state updates and trade-offs between incremental state transfer of operators and the initialization of operators from historic event streams. The problem of state transfer relates also to the problem of reliable event processing (cf. Chapter 6),



(a) Boundaries (b) Coordination

Figure 5.3: The figure shows the issues in achieving spatial-temporal consistent event streams.

namely when keeping redundant operator instances or recovering the operator state after failures. Adaptive CEP [114] and Transition capable complex event processing (TCEP) [70] also deal with the problem of changing the behavior of a CEP system dependent on the user context by dynamic changes to the required QoS. Those works explore a method for exchanging the underlying mechanisms of the CEP system as part of a *transition* [4]. Therefore, dynamic operator migrations can become necessary. TCEP aims to minimize the migration cost that results from such a transition of placement mechanisms.

5.3 CONTRIBUTIONS

In this section, we summarize contributions of our own on Mobile Complex Event Processing (MCEP) [81]. MCEP allows processing events in a spatial-temporal order imposed by dynamic updates of context restrictions. In difference to traditional moving range queries, a context restriction in MCEP is defined over all dependent processing steps of an operator graph. The events are delivered according to a consistency model that enforces an order of events along two dimensions: a temporal dimension (for events of each valid context restriction) as well as a spatial dimension (following the sequence of updates of context restrictions). As part of MCEP we have proposed methods to dynamically reconfigure the operator graph in order to comply with this spatial-temporal consistency model. In this way, the reconfiguration capabilities of MCEP enable applications to describe with high precision their context restriction. Furthermore, we have proposed methods for dynamically migrating operators in order to minimize the resource usage in a non-disruptive way and account for latency constraints of a consumer and bandwidth efficiency.

In more detail, the underlying consistency model accounts for a sequence of range updates $R_1, \ldots, R_N$, each valid for a sequence of time spans $\Delta_1, \ldots, \Delta_N$. A *spatial-temporal consistent* execution delivers for each range R_i, all events that can be detected over windows W_j which overlap with the time span Δ_i for which R_i was valid. The order of those events respects the temporal order of the timestamped events. Moreover, the spatial-temporal consistent execution ensures that the order of events respects the order of range updates, i. e., if an event e_i is produced for R_i and e_j is produced for R_j then e_i is delivered ahead of e_j if $i < j$.

For example, in Figure 5.3a the time span Δ_{i+1} for which R_{i+1} becomes valid is determined for $\omega_{\rightarrow}$ by $start(R_{i+1})$ and $end(R_{i+1})$ which corresponds to the time of occurrence for the range updates R_{i+1} and R_{i+2}, respectively. Since $W_{i+1,1}$ and $W_{i+1,n}$ overlap with Δ_{i+1}, a delivery in temporal order is ensured for all events produced with respect to the windows in $[W_{i+1,1}, \ldots, W_{i+1,n}]$.

Note that detecting all events of the partially overlapping window $W_{i+1,1}$ can require to process events ahead of $start(R_{i+1})$. For example, the preceding operator ω_a has to produce all events comprised in $W_{i+1,1}$. In order to detect all necessary events each operator determines in MCEP an extended time span dependent on the window boundaries of its succeeding operators. For each operator ω_j of the operator graph, MCEP determines a minimal offsets $(start_j, end_j)$ to extend the operator's time span appropriately. In consequence, each operator receives from its predecessors exactly those events that it requires to process for producing a spatial-temporal consistent stream. When an operator has produced all events with respect to a range R_i, it will send a marker message separating the stream between two ranges (cf.Figure 5.3b). The marker, therefore, helps for all succeeding operators, including the consumers, to understand that all subsequent events need to be interpreted with respect to a new range. In response to a marker message, an operator completes processing its window, forwards all produced events to the successor and collects from its successor the offset boundaries of the new range. Based on these boundaries the operator can determine its own offset and notify its predecessors on the offset values valid for the next range.

The efficiency at which reconfigurations can be performed depends on the concrete window semantics and the synchronization overhead. For instance, certain window semantics like a fixed-size sliding window would allow skipping the collection phase, but determine the offsets locally by interpreting the successor's windows semantics. Moreover, the reconfiguration can benefit from the spatial overlap of ranges, reducing the number of events to be transmitted to fill the first windows. This yields a bandwidth reduction and reduces at the same time the latency until the first events can be detected right after a range update.

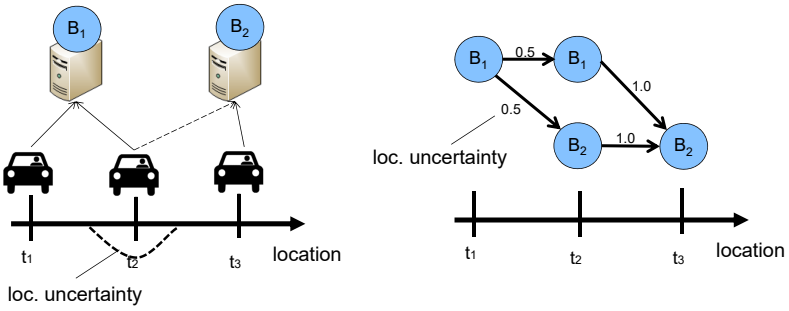


Figure 5.4: In this figure the uncertainty about the access to the network is captured in a Markov model.

In order to improve the placement of operators under the dynamics of mobile consumers and context restrictions, we have proposed, as part of MCEP, mechanisms for the dynamic migration of operators. To minimize the resource usage (e. g., processing resources and bandwidth) and comply with QoS constraints of the consumer (e. g., latency restrictions), the migration approach accounts for the state of the operators, i. e., migration points are selected which minimize the state that needs to be transferred. Furthermore, to avoid downtimes of the operators, necessary state like context information and duplicated events needs to be transferred ahead of the actual migration time.

In consequence, the target to which the operator will be migrated needs to be decided under uncertainty of the location. Building on the mobility models of consumers, MCEP maintains a model that describes for a specific time the probability that a node will access the network through a specific broker B_i. In Figure 5.4 the mobility model shows how the car is moving over time. In particular, the connectivity to a broker changes because of the car's mobility. At t_1 it is certain due to the car's communication range that the car will connect to the network via B_1. Similarly, at time t_3 the car will connect to broker B_2. However, at time t_2 the car may access the network either via $B1$ or B_2 since both brokers are in communication range. MCEP models this dynamic behavior as a Markov process, where each state of the Markov process corresponds to a configuration of how the consumer accesses the network at time t_i. For example, in Figure 5.4 there exist two states for time t_2, each reachable from the state (B_1, t_1) with probability 0.5. Each such configuration can have a different optimal placement of operators, and each imposes a cost for migrating the operators when transiting with a probability from one state to the next one. Hence, solving the placement problem for each state of the Markov Process model and accounting for the transition probabilities of states, leads to a graph in which edges correspond to changes in the placements of operators. The weights of edges define the resource

cost imposed by changing to a different placement including the cost of migrating operators.

MCEP provides methods to determine the best migration path from a sink that corresponds to the current location to a destination defined by the mobility model. The cost for determining the migration path depends on the placement mechanism and the number of states that are considered in the Markov Process model. For example, if the mobility model imposes high uncertainty, the event processing system may restrict the scope for which migrations are planned ahead of time reducing the number of states as well as coarsening the time steps to be considered. The event processing system may also build on distinct placement mechanisms which require different time to converge, but also differ in the quality of their solution. A dominant factor for the migration cost is the overall state to be transferred. If this state varies faster than the time steps of the Markov model the state transfer can be postponed (this is the typical behavior observed in our evaluation studies). An alternative is to model the cost dependent on the operator's internal state anticipated at time t_i.

In order to realize the state transfer MCEP builds on recovery mechanisms we will discuss in Chapter 6. For a detailed evaluation of the trade-offs of the discussed design decisions in the context of a traffic scenario we refer also to [83] and [61]. Moreover, in [82] we extended the reconfiguration and migration approach of MCEP to account also for overlapping context restrictions of multiple consumers.

6 OPERATOR RECOVERY

In the previous chapters, we observed that operators of the operator graph are executed in a distributed environment. Moreover, we observed that the set of detected events and the order in which events are delivered depends on how reconfigurations and dynamic placement of operators are performed. When failures occur, event processing is highly susceptible to out-of-order delivery of events as well as false positive and false negative detection of events. In particular, node failures are a major cause that can lead to situations where operator state is entirely lost.

In this chapter, we analyze and propose mechanisms that support the consistent processing of event streams in situations where the hosts of operators can fail and lost operator state must be recovered. Such mechanisms impose additional overhead for an event processing system in two main dimensions: i) at run-time to prepare for failures, and ii) at recovery time to respond to failures. Therefore, a central objective is to minimize the overall recovery overhead and find a good trade-off between preparing for failures and the recovery of operators state. This chapter builds on the following contribution:

F6: Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. "Rollback-recovery without checkpoints in distributed event processing systems." In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 27–38. DOI: 10.1145/2488222.2488259

6.1 THE RELIABLE EVENT PROCESSING PROBLEM

While the placement of operators on multiple distinct processing resources helps increasing the performance of an event processing system, it also causes event processing systems to become more susceptible to failures. The failure, e. g., a crash, of a single node hosting operators can cause the operator state to be lost and in this way render the produced event stream useless in the form of false negative or false positive (e. g., for the "not" operator) detection of events. The probability of failures raises with the number of distinct non-replicated nodes over which the operator graph is distributed. Wherever false positives and false negatives are not acceptable an event processing system must be complemented by additional reliability mechanisms. In the reminder of this chapter, we will concentrate for the sake of simplicity on failures according to the fail-stop failure model, i. e., in

response to a failure a failed node hosting operators will stop doing any computational work and stop sending messages. For dealing with limitations of this failure model we refer to well-known complementary reliability mechanisms, e. g., using stable storage for crash recovery [62] or building on cryptographic signatures to deal with Byzantine failures [34].

In the situation of a crash-stop failure, the event processing system has to be able to respond to recover the lost operator state based on information available on other nodes or communication channels in order to avoid a false negative and false positive detection of events. Research in distributed systems proposed a wide range of mechanisms such as replication [96], error recovery and compensation [89], failure detection [26], and timestamping [66, 99] that can be composed to respond to failures. The way those mechanisms are composed depends on the concrete consistency specification at hand. The consistency specification determines the guarantees, but also the overhead needed to enforce them. Note, that we have already stated a strong consistency specification for dynamics in former chapters. We can adopt this specification for the situation of node failures, i. e., a *strong consistent execution* will ensure, in the presence of node failures, the stream of produced events to be indistinguishable from an execution without failures. In more detail, we require from the event processing system to ensure for an observed set of input streams I the following properties:

- *Completeness:* The event processing system eventually detects all events for I, i. e., no false negatives.
- *Correctness:* The event processing system does not generate new events for I, i. e., no false positives.
- *Order:* The events can be processed in a deterministic order, respecting their temporal timestamps.

The consistency specification could be relaxed, e. g., enforcing the order of events only eventually, possibly reducing the overhead of ensuring reliability at the cost of intermediate false positives and negatives.

In this chapter, we focus on a strong consistency model for event processing which applies to situations were false positives and negatives are not acceptable. In order to ensure strong consistency, we address two key problems. We have to ensure that each operator works on its incoming streams in a deterministic order under failures. This requires concepts to detect disorder of events, e. g., by appropriate time stamping mechanisms and cope with the late arrival of events. Moreover, we have to consider concepts for recovering lost state. This allows for two design decisions, the replication of operator state and the recovery of operator state from checkpoints.

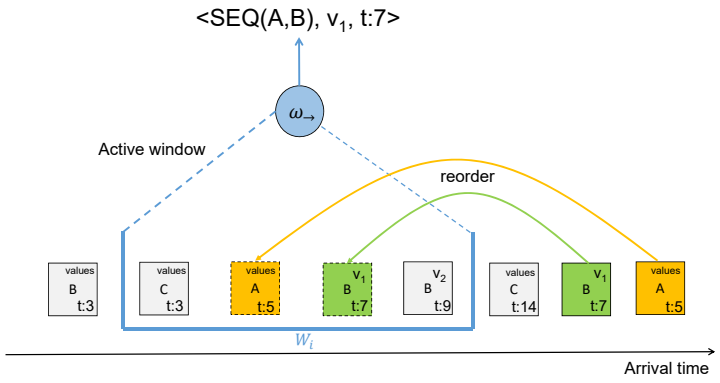


Figure 6.1: This figure illustrates the difference between arrival and processing order of events.

6.1.1 *Completeness and order of event streams*

The events that can be detected by an operator depend on the set of observable incoming events. Let W denote the window restricting the events to be processed in a processing step, then the correct detection of an event pattern requires that all events within W are available (completeness) and that the events can be processed in a deterministic order (total order). For example, Figure 6.1 illustrates several problems that can occur in an event processing system due to disorder, late notifications, or loss of events. In the example, the arrival of events $\langle A,\ t:5\rangle$ and $\langle B,\ v_1,\ t:7\rangle$ does not follow the order of events imposed by t. Without further mechanisms the situation before receiving $\langle B,\ v_1,\ t:7\rangle$ is indistinguishable from a situation were both events did not occur or simply due to failures the events were lost. The operator may not be able to detect the sequence, or even detect a sequence comprising a different value v_2 instead of v_1 comprised in $\langle B,\ v_1,\ t:7\rangle$.

Although the example focuses only on a single window and operator instance, it is important to note that in a distributed event processing system the processing of events happens over multiple dependent windows and several operator instances, e. g., replicas, which have to agree on the same set of events and their order. In particular, we require the following reliability properties:

- *R1:* Each produced event of a correct source will be eventually processed by all correct operator instances with a matching input stream.

- *R2:* Each event that has been processed by a correct operator instance for a window W will also be contained in any other processed window W' with a matching input stream.

- *R3:* If e_i and e_j are events, the events will be processed in the same order by all windows W with matching subscriptions.

Property *R1* ensures that once an event e_i has been admitted by the event-based system, all correct operator instances will be able to access e_i in all relevant processing steps. Property *R2* is important to account for faulty sources. Before such a source is detected to be faulty, events still can be admitted to the event-based system. Property *R2* will ensure once an event is admitted, it will be consistently processed by all dependent operator instances. Finally, Property *R3* ensures that events can be processed in deterministic order.

In realizing properties *R1–R3* multiple brokers and sources must agree on the set of admitted events and their order. For efficient processing, events can carry in addition to the temporal timestamps determined at each source and operator, logical timestamps that allow identifying missing and unordered events in the stream. Additionally, markup events generated by event producer may indicate a time interval for which no events will follow and help succeeding operators to order incoming events from distinct producers. Furthermore, the temporal timestamping mechanism for produced events is required to be deterministic, i. e., it is a deterministic function over the temporal timestamps carried by events of the incoming streams, e. g., taking the min/max temporal timestamp of events imposing an event pattern.

In summary, this requires suitable coordination and timestamping mechanisms and

1. methods for efficiently ensuring a deterministic ordering of events,
2. methods for ensuring completeness of the event stream,
3. methods for ensuring the availability of events until all events that depend on the event have been detected.

6.1.2 *Recovery from node failures*

Even if all produced events can be reliably accessed, i. e., all produced events of a stream I can be processed in the same order and are completely transferred, failures of nodes hosting operators can lead to the situation that the processing state is lost and in consequence not all events that could be detected will also be produced. In case of a node failure, the operator needs to be recovered in a way that it will produce the same sequence of events as in a failure free execution.

For performing a recovery, there exist two basic design decisions. For the first design decision —sometimes referred to as replicated state machine— the operator's state is replicated on multiple nodes. All replicas take as input the same deterministically ordered event

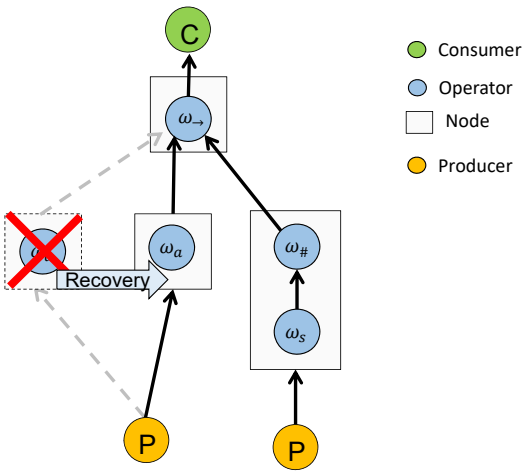


Figure 6.2: This figure shows the recovery of an operator upon a failure.

stream and as a result produce the same events. Therefore, the run-time overhead raises with each replica in terms of redundant processing overhead as well as in terms of bandwidth consumption for redundantly transmitted event streams. An alternative way of achieving reliable processing of events is to perform checkpointing, i. e., periodically store the state of an operator in the form of a snapshot. Upon a failure the snapshot can be recovered on any node. After the snapshot is recovered, the node needs to replay all incoming events from the checkpoint at which the snapshot was taken. In a failure-free execution, the run-time overhead can be controlled by the frequency of snapshots and the point in time the snapshot is taken. Recall from Chapter 5 that the internal state depends on the processing state of an operator. Ideally, a snapshot avoids any internal state to be transferred to other nodes. The checkpointing methods, on the other hand, increases the recovery time depending on the frequency at which snapshots were taken and the size of the state that needs to be recovered.

When applying methods for checkpointing, it is important to notice that the recovery of events depends on the processing results of preceding operators. Therefore, recovery mechanisms for checkpointing need also to account for multiple dependent failures when recovering an operator's state.

Overall, dealing with recovery requires to deal with

1. methods for minimizing the runtime overhead of event processing and accomplishing acceptable recovery times,

2. methods for coping with dependent failures in the recovery of operators.

6.2 RELATED WORK

Originally, work on event processing focused on the execution models and the expressive specification of events [2, 25, 41]. The original design decision was to host the logic of detecting events on a single node. Therefore, ensuring reliability did initially not require specific mechanisms beyond well-known distributed systems mechanisms for active and passive replication [62, 96]. Nevertheless, in developing query languages for such systems, important foundations for consistent event processing were made. For example, the Snoop language [25] and later Tesla [30] have identified very fundamental concepts that allow modeling the execution steps of most event processing systems (cf. Chapter 2), like selection policies for events, and consumption policies to evict events from a window. Building appropriate execution models are a prerequisite for our contribution to understand when and how to replicate an operator state.

With the emergence of distributed event and stream processing, research early on accounted for mechanisms to respond to failures. Early work noted that responding fast to failures can be accomplished by *relaxing the consistency requirement* [9, 54]. As part of the CEDR (Complex Event Detection and Response) [12] language the potential of trade-offs between weak consistency models for improving the performance are discussed. Balsas et al. [9] proposed an eventual consistency model in which operators also produce events even if not all input streams are accessible, e. g., due to node failures or unreliable or slow communication links. Consumers will receive a consistent event stream only if all operators of an operator graph receive a complete event stream. By annotating events whether they are produced on complete (stable) or incomplete information (failure), also stronger consistency specifications can be realized. Then only those events are forwarded which are stable. Jaques-Silva [54] proposed a language extension for the SPADE language of System S [42] that allows defining operator-specific recovery mechanisms. The proposed consistency specification also follows from the weak consistency semantic we discussed in Section 6.1.

Very similar considerations have been taken without explicitly focusing on node failures, but dealing with the effects of out-of-order arrival of events [21, 67, 78]. Incompleteness is caused by the disorder of events due to variations in event transmission times [21, 67] or communication errors [78]. In an asynchronous system, long transmission times and failures of nodes are known to be indistinguishable [40]. Therefore, these approaches define a skew an operator needs to wait for resolving disorder of events. Late arrivals beyond the skew can still cause inconsistencies. Acceptable disorder and latency are an application-specific trade-off. In addition, those approaches provide mechanisms to compensate effects once disorder of messages is ob-

served, e. g., by aborting transactions when observing out-of-order events and then restoring the state of the operator to a previous state where disorder can be resolved. Mutschler et al. [76] proposed mechanisms to adapt the skew dependent on the CPU load in order to optimize the performance gain of speculation and minimize the cost for compensating for disorder.

In order to efficiently support the correction of incomplete processing steps or the reinitialization of operator state, the concept of *event histories* has emerged. Already Borealis [1] offered methods for time travel, i. e., to process historical events, and Dindar et. al [33] propose as part of the Pattern Correlation Queries (PCQs) semantics how to improve the efficiency of event processing by historic access. Also, publish/subscribe systems like PADRES [53] have proposed abstractions to access historical event steams. We also build as part of Mobile CEP (cf. Section 5.3) and the safepoint recovery method (cf. Section 6.3) on the idea of distributed event histories to initialize operator state. In addition, we account for minimizing the number of events to be preserved in the event history dependent on the progress of the event processing system.

Many works aiming for *strong consistency* assume —in many cases implicitly— that the nodes interact with respect to a synchronous communication model, i. e., there exist upper bounds on message delays, failures can be detected reliably, and events arrive in total order. However, it is important to note that some of these strong assumptions on synchrony can be weakened dependent on the location of the sequencer performing message ordering. For example, Bhola et al. [15] notice that given reliable sources and an acyclic dataflow, the total order property can be preserved in subsequent routing steps even if brokers communicate with respect to an asynchronous communication and can fail silently (unreliable failure detection). This principle has been generalized for reliable publish/subscribe [69, 117] and reliable event processing [111] to enhance scalability. Thereby, the main synchronization effort for reliable total ordering is kept local for each data source and event producer, while the subsequent distributed processing of events of distributed brokers can be performed scalable without strong synchronization of brokers.

In accomplishing strong consistency for event processing under failures, approaches build on active replication and checkpointing, or a combination. Active replication approaches [50, 111] keep for every operator a set of k replicas. The replicas are placed on nodes, such that for a maximum of $f < k$ node failures, the operator graph has for every operator at least one correctly working replica. Therefore, replication involves significant run-time overhead in form of redundant processing steps and messages. Voelz et al. [111] show that the message complexity for replication can be bounded linearly with the

number of replicas if synchronous communication is required only at the sources.

Most systems aim to avoid or limit the use of active replication and build on continuous checkpoints [64] to recover operator state. Checkpointing the state of the operators reduces the run-time overhead caused by redundant processing. Nevertheless, checkpointing mechanisms require to take a snapshot of the operators state and replicate this state. Compared to active replication, longer recovery times are typically needed to initialize the operator state at another node. Initialization can require additional overhead to transfer the checkpoint to the new target destination. In upstream backup [39, 49] checkpoints and additional state to recover unacknowledged events is stored at preceding operators. However, the checkpoint is lost when multiple dependent failures occur. The obvious improvement, namely replicating the checkpoint at multiple redundant locations, however, results in a significant transfer cost. Therefore, the proposed safepoint recovery method [60] (cf. Section 6.3) builds on an execution model to understand when an operator has completed a processing step, and initialization can be performed only from replicated historic events without checkpointing the operator's state.

Subsequent work has also explored trade-offs between using replication and checkpointing. For instance, Heinze et al. [46] propose to adaptively transit between mechanisms for replication and checkpointing. Wermuth [36], studied reliable event processing in highly dynamic environments where nodes communicate in an ad-hoc manner. In particular, this work proposes a more fine-grained operator state model than the proposed safepoint recovery. It accounts for intermediate processing steps and allows the combination of distinct recovery mechanisms.

Even stronger abstractions beyond the introduced consistency model has been proposed in the context of transactional stream processing [19] and transactional event processing [3].

6.3 CONTRIBUTIONS

In this section, we summarize an own contribution to the *safepoint recovery method* that ensures for a distributed execution of the operator graph a consistent detection of events in the presence of multiple dependent failures [60]. Contrary to traditional checkpointing, the proposed *safepoint recovery* avoids the transfer of operator state by performing checkpoints only when the internal operator state is minimal. This allows both reducing run-time and recovery overhead.

In order to avoid the cost of checkpointing the entire operator state, safepoint recovery detects points in the execution of the operator when an initialization of the operator state can be performed from an operator's intial state by only replaying events of a stream, we sub-

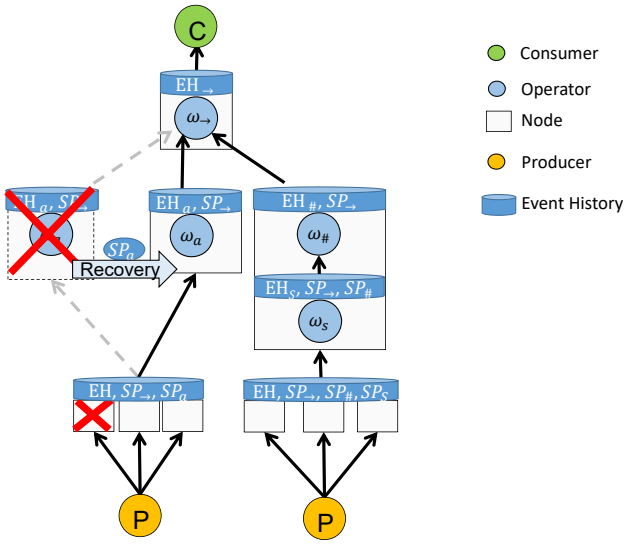


Figure 6.3: This figure shows the recovery of an operator upon a failure.

sequently refer to as *historic events*. This is accomplished —similar to Chapter 4— by an execution model that captures expressive window semantics of an event processing system. The execution model is used to detect so-called *safepoints* in the event stream from which on a freshly initialized operator will produce a strongly consistent sequence of events. In addition, safepoint recovery proposes how to replicate minimal state to support fast recovery and sustain multiple dependent node failures.

Safepoint recovery builds on a distributed f-reliable *event history* $EH_I(t_{start}, t_{end})$ which allows retrieving for a stream I and a temporal range $[t_i, t_j] \subset [t_{start}, t_{end}]$ all produced events in deterministic order, respecting the order imposed by the temporal timestamps. The event history maintains for all events a logical timestamp ts. In particular, an event e_k with timestamp $ts(e_k)$ is a direct predecessor of e_l with timestamp $ts(e_l)$ if $ts(k) = ts(e_l) + 1$. Consequently, gaps, duplicates, and disorder of events can be detected by each receiver by comparing the logical timestamps of the received events.

Figure 6.3 illustrates how the event history abstraction is coupled with the event processing logic. The event history for each stream can be realized in two different ways. The $EH_S(t_{start}, t_{end})$ consists of $f + 1$ replicas, in particular, the primary events are replicated to sustain f failures. Alternatively, EH_S can recover its state by reproducing the events and their logical timestamps from information at its predecessors. For example, in Figure 6.3 the state of EH_a is recovered by reprocessing events from replicated primary events. An additional failure, of $\omega_{\rightarrow}$, $EH_{\rightarrow}$ could be recovered by reprocessing at $\omega_{\rightarrow}$ events from EH_a and $EH_{\sharp}$ starting with the timestamps stored in the safepoint $SP_{\rightarrow}$.

In consequence, a safepoint[1] SP_ω for operator ω consists of two parts. First, it comprises the logical time $ts(SP_\omega)$ when the safepoint was captured, i. e., the event to be produced after recovery at SP_ω will be $ts(SP_\omega) + 1$. Second, for each incoming stream of ω, say I_j, $SP_\omega[I_j]$ comprises the timestamp and point in the stream for which ω is guaranteed to produce a consistent event stream. To avoid loss of safepoints during failures, a safepoint itself is replicated at least at $f + 1$ nodes hosting preceding operators or replicas of the event history. In addition, safepoints of the preceding operators need to be able to reproduce the event history of incoming streams. For example, let ω' be a predecessor of ω for incoming stream I_j, then a safepoint at time $t \leq SP_\omega[I_j]$ must be available at preceding nodes to restore the event history $EH_{\omega'}$.

New versions of a safepoint of an operator can be added whenever the execution model detects that the operator state can be reproduced from a new safepoint. A new version is installed if it is acknowledged by $f + 1$ replicas. Similar to checkpointing, the frequency in adding new versions of safepoints can be traded for the effort in recovering from failures, i. e., the lower the frequency at which versions are installed the higher will be the recovery effort. However, unlike for checkpointing, the installation cost of a version by transferring only logical timestamps is significantly reduced. Old versions of ω are deleted if all safepoints depending on ω can be restored by a more recent version of the safepoint. In addition, e in EH_ω can be deleted on a node when $ts(e) < ts(SP_\omega)$ for the oldest installed version of SP_ω.

In order to detect that a safepoint can be captured, the safepoint recovery builds on the following dependencies and behavior of the operator execution environment. The operator execution environment is assumed to comply with the following execution model (similar to the execution model proposed in Section 4.1):

1. The operator ω works on a restricted part of the event stream defined by a window W that determines the events to be selected from its incoming streams I_j.

2. To find a specific event pattern described as part of an event correlation language like Snoop [25] the operator executes the logic f_ω that triggers a complex event whenever the pattern is contained in W.

3. After a processing step (detecting or not detecting an event) the operator execution environment selects a new window W' that comprises events of W or events with larger timestamps than those contained in W.

1 note, that we use here a more compact description of a safepoint then originally performed in [60]

Any produced event needs to be captured by the safepoint recovery method, timestamped, and inserted into the event history. Furthermore, whenever an execution step over a window is completed, safepoint recovery needs to be notified by the operator execution environment about the consumption of events from W. This allows keeping track for each incoming stream I_j of the events with the smallest timestamps comprised in W. Therefore, after every execution step it is possible to determine $SP(\omega)$ by the timestamp of the last produced event, and $SP_\omega[I_j]$ as the lowest timestamped event which was not consumed from W. The safepoint will yield a strongly consistent event stream for all parameter contexts of the Snoop language, which is known to be one of the most expressive correlation languages. Nevertheless, the model will not capture all possible consumption policies which could be defined in theory. That would require to store not only the first evicted event as part of the safepoint, but also preserve all consumed events between two versions of a safepoint. Finally, the granularity at which safepoints can be taken depends on the complexity of the correlation function. Since the intermediate state of the correlation function is not considered long processing steps will also impose long recovery times. For such correlation functions preserving intermediate state and snapshots of operator state can still be worthwhile and may yield overall lower recovery times.

7 EVENT ACCESS CONTROL

Event-based systems interconnect many consumers and producers and are dynamically deployed over heterogeneous nodes and domains. Not all entities including the nodes managing the broker network can be trusted in the same way, but typically require specific privileges for the access of events. Without corresponding security mechanisms for administrative domains to control security privileges, event-based systems are intrinsically vulnerable to unauthorized entities injecting unauthorized events, as well as declaring themselves as legitimate consumers of events. Integrating appropriate security mechanisms, which i) enable the management and control of security privileges and ii) preserve at the same time important properties of event-based systems, i. e., the decoupling of producers and consumers, efficient routing of events, and scalability, is a very important, but also highly challenging task.

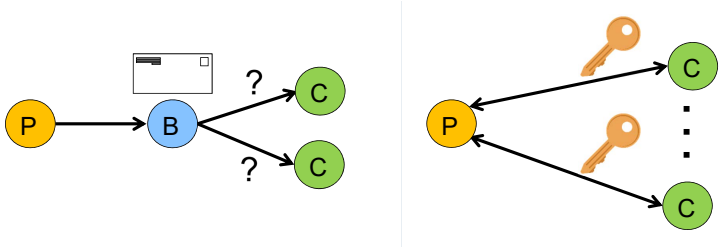
In this chapter, we focus on problems related to confidential content-based event distribution and scalable key management. In particular, we introduce mechanisms that can be used to control the access of events with respect to the content-based subscription model and counteract security challenges of event-based systems by a combination of methods from pairing based cryptography and the spatial indexing model we introduced in Chapter 3. This chapter builds on the following contribution:

F3: Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. "Securing Broker-Less Publish/Subscribe Systems Using Identity-Based Encryption." In: *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2014, 25.2, pp. 518–528. DOI: 10.1109/tpds.2013.256

7.1 THE PROBLEM OF SECURING EVENT-BASED SYSTEMS

In an event-based system, there are two major security requirements. First, an event-based system must enforce that only events produced by authorized producers will be accepted by interested consumers. Second, the event-based system needs to ensure that consumers can only access the content of events to which they are authorized.

In addition, an event-based system should enforce privacy by protecting subscriptions and advertisements, i. e., the event-based system should be able to deal with adversaries that are eager to learn the interest of producers and consumers. Furthermore, the rights management used to enforce security and privacy policies should allow



(a) Routing (b) Key exchange

Figure 7.1: The figure illustrates key problems related to confidential routing and scalable key management

for flexibility, i. e., allow for dynamic subscriptions and unsubscriptions, new advertisements, and granting new or revoking privileges for producers, brokers, and consumers. Finally, the efficiency in routing events, bandwidth-efficiency, and end-to-end latency should be affected as little as possible.

To ensure confidentiality and authenticity when exchanging events over a non-trusted infrastructure, the content of events need to be encrypted, and events must carry signatures. Encrypted events pose a severe challenge for efficient content routing since in-network filtering mechanisms for accomplishing bandwidth-efficiency typically perform routing decisions exactly based on the content of an event (cf. Figure 7.1a). Moreover, every new subscription or advertisement can impose significant overhead in key management to enable producers, brokers, and consumers encrypting content and verifying the authenticity of events. For example, introducing a new advertisement overlapping with $|S|$ subscriptions would require with standard asymmetric cryptography a significant number of $O(|S|)$ keys and at the same time violates the requirement of decoupling producers and consumers (cf. Figure 7.1b). In the following, we explain these problems in the context of *confidential event distribution* and *scalable key management*.

7.1.1 *Distribution of encrypted events*

Recall that the basic idea of content routing is to offer bandwidth-efficient routing by filtering events early and possibly even performing additional event processing on intermediate brokers. Therefore, every intermediate broker requires privileges for subscribing, processing and publishing events. Given that event-based systems can interconnect producers and consumers from distinct networks and domains and brokers may utilize heterogeneous resources from multiple distinct stakeholders —for example in the Internet of Things over domains of devices, network operators and operators hosted at a data center— not all brokers should hold the same privileges, but

only those trusted in their specific resource environment. For example, consider a business process spanning two companies C_A and C_B. Internal business events of C_A and C_B will only be processed on resources in control of C_A and C_B, respectively. Nevertheless, some events of relevance to the business process must be forwarded to consumers of both C_A and C_B from brokers both residing in the resource domains of C_A or C_B. Even within the domain of the company, there may exist several subdomains and distinct resource environments that should be isolated. For example, the manufacturing processes should be shielded from the business processes and vice versa.

In turn, brokers with distinct security privileges will need to cooperate for establishing efficient forwarding paths. This requires from brokers to exchange reachability information which can be used to infer knowledge on subscriptions and advertisements of connected consumers and producers. The kind of reachability information to be exchanged also depends on the use of the cryptographic method and how credentials are distributed in the broker network.

In summary, we are looking for methods that

- allow to efficiently routing events to authorized consumers via brokers with heterogeneous security restrictions,
- minimize knowledge that adversaries can infer on subscriptions and advertisements.

7.1.2 *Scalable Key Management*

In an event-based system, the key management must ensure the decoupling between producers and consumers. Similar to brokers enabling a decoupling of producers and consumers in forwarding events, secure event-based systems require a key management service that offers producers and consumers keys that allow to encrypting/decrypting the content and signing/verifying the authenticity of events. For a producer and a consumer the ability to perform efficient security operations depends on the frequency at which new keys need to be obtained, the number of keys and the corresponding number of cyphertexts that need to be generated in forwarding events.

Similar to the management of a broker topology, the number of installed keys should only be changed if the credentials of producers, brokers, and consumers change, i. e., keys are no longer valid or are revoked by the key management. The effort for the key management depends on the set and number of keys that need to be installed at a consumer and producer. Ideally, this effort should not depend on a global view on the set of installed subscriptions and advertisements, but rather on the credentials needed for producers to efficiently encrypt, decrypt, sign and verify the content of events. Ideally, key management allows consumers generating as many keys as possible

locally and in this way minimize the transfer and generation cost of keys.

This requires to select appropriate cryptographic paradigms that allow to encoding and decrypting events with minimal number of keys. At the same time the way ciphertexts are generated needs to be compliant with content routing forwarding schemes allowing for local routing decisions of each of the brokers.

In summary, we are looking for methods for key management that

- preserve the decoupling of producers and consumers,
- support fine-grained restrictions on the content of events,
- minimize the number of keys to be maintained at the brokers,
- allow for dynamic key management.

7.2 RELATED WORK

Wang et. al. [113] originally introduced and analyzed important security goals for realizing content-based publish/subscribe; they especially motivated central problems regarding routing on confidential content and preserving the decoupling of producers and consumers.

Approaches for securing content-based routing typically address i) secure event distribution techniques in preserving confidentiality of events and ii) models for offering privileges and corresponding cryptographic keys for producers and consumers to access and process events. The two issues are often highly interwoven. In particular, approaches concerned with confidential event distribution build on different models of how the broker network can be trusted. For example, in the role-based model to access events [14], the brokers are trusted and will only accept advertisements and subscriptions of producers and consumer if they can be authenticated with respect to a specific role. The trust level of brokers can be organized in multiple domains [85]. In addition, broker groups often build on a common group key to ensure the confidential transmission of events [80]. Also, the granularity at which key management is performed varies. PS-Guard [102] proposed for a trusted broker environment a fine-grained hierarchical key management accounting for ranges of attribute values by building on group-based key exchanges. A consumer covering with its subscriptions k distinct value ranges of the hierarchy, therefore needs to obtain from the key server k distinct private keys. In our proposal building on an identity-based encryption scheme [110], we also follow a fine-grained hierarchical key management building on the spatial indexing method that allows to establishing privileges based on the level of ranges for attributes. The identity-based encryption scheme, however, reduces the exchange of private keys, i. e., only a single private key per subscription and advertisement needs to be

exchanged with a key server instance. Remaining public keys can be generated locally by each producer, consumer, and broker. Therefore, the cost for rekeying with the key server can be significantly reduced.

To limit information leakage of content-based publish/subscribe Raicu and Rosenblum [88] propose encrypting only confidential content, but using other (non-confidential) attributes or even additional attributes that can be used for the efficient routing of events. Similar techniques have also been proposed later in combination with fine-grained access restrictions in EventGuard [103], an extension of PSGuard. In contrast, with the broker network in the identity-based encryption method (cf. Section 7.3) one can define any arbitrary trust levels. This allows, for instance, realizing the broker functionality only by consumers themselves without requiring specific trusted brokers.

Onica et. al. [79] surveyed alternative approaches in providing confidential event-routing and compared the overhead imposed on key management. In particular, two complementary alternatives to the proposed identity-based encryption method were introduced: i) asymmetric scalar preserving encryption [27] and ii) homomorphic encryption schemes [77]. A possible advantage of those encryption schemes is that a broker does not even need the ability to understand what is the content and still can perform computational operations on the content, e. g., adding two encrypted values or performing a filter operation. Such operations, e. g., the filter operation, can therefore be used to perform bandwidth-efficient routing decisions. Also, the type of operations are not only limited to filtering operations but can also support more complex operations of event processing systems [101]. Nevertheless, building on these schemes or even combining the strength of the encryption schemes with identity-based encryptions is known to impose additional cost in key management [79] and overhead for encryption/decryption [73]. This imposes at the current state of the art limitations on the practical applicably of full homomorphic encryption schemes regarding achievable throughput and end-to-end-latency.

Extending trust models for broker-based networks in event processing systems raises further issues. Schilling et. al. [95] note that the transformation of events can allow also unauthorized consumers to access events by using inference techniques on the complex events. Therefore, the authors propose an access control scheme that mitigates this problem by allowing producers to annotate events with access policies. Access policies define an acceptable level of inference threshold for which other brokers and consumers may access events. The brokers therefore need to monitor the inference probability and consolidate policies when forwarding events according to the installed access. Palanisamy et al. [84] refine this idea and propose specific mechanisms to enforce privacy constraints of consumers by

explicitly reordering events dependent on the privacy requirements at hand.

Beyond efficient key management as well as preserving confidentiality of events in the scope of this thesis, establishing trust relations between the engaged entities is an important and worthwhile research direction. A step in determining and controlling, is the work of Dwarakanath et al. [35] on automatically deriving trust relationships from the social context of the users. Another important future direction addressed recently, is the use of secure computational enclaves, for example, supported by the Intel SGX hardware. Initial research findings have investigated routing and key management [75]. Also here identity-based encryption may be a good fit for scalable fine-grained key management and at the same time facilitate integrating heterogeneous trust in entities and domains of an event-based system.

7.3 CONTRIBUTIONS

In the following we detail a solution of our own on scalable key management that can be used to accomplish fine-grained content-based filtering [110]. The proposed solution preserves the decoupling of producers and consumers, while limiting the number of keys to be exchanged. Furthermore, the approach offers efficient means to distribute events as well as to constrain in a flexible manner the duration for which producers and consumers can access events.

The approach for ensuring confidentiality builds on *pairing-based cyptography*, and a key exchange principle refered to as *identity-based encryption*, which was originally described in [100]. In identity-based encryption, a message can be encrypted with any string identifying the content to which one or multiple recipients obtain credentials.

The underlying principle of using identity encryption is illustrated in Figure 7.2. Assume Alice wants to send a confidential message to Bob, identity-based encryption allows to generating a cyphertext with a master public key and a string comprising the email address of Bob. In this way Alice does not require to exchange with Bob a specific public key, but only uses knowledge about the identity of Bob, e. g., performing a lookup in her local address book. The key management service, controlled by a logically centralized key server, can generate for Bob's email address a corresponding private key, which will allow Bob to decrypting the message of Alice. Similarly, if Alice decided to send a message to a mailing list at a remote service, the key server can distribute to all users including Bob credentials for the name of the mailing list in form of a single private key. This way sender and receiver remain decoupled by the key server functionality and a sender is not required to exchange public keys with each possible recipient.

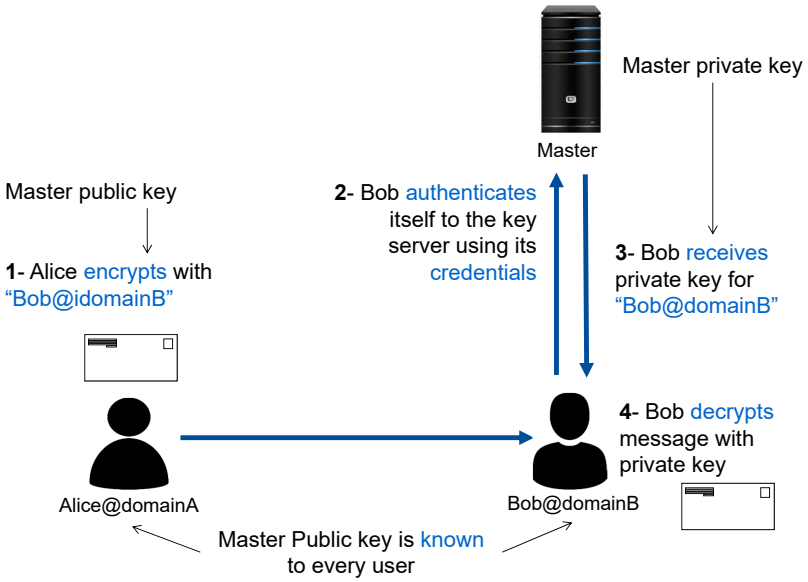


Figure 7.2: In this figure the basic principle of identiy-based encryption is explained.

As a next step, we show how the principle of identity-based encryption in combination with the spatial indexing approach Section 3.3 can be used to accomplish confidential event routing and ensure that only authorized producers will be able to send events.

Producers and consumers can obtain credentials from the key service for advertisements of producers and subscriptions of consumers. It is important to note that the logically centralized key service functionality may be physically distributed similar to the broker functionality of an event-based system. A key server owns the master key and decides —based on the access policy specifications at hand— how producers, brokers, and consumers can access events by granting corresponding credentials. Each credential obtained by a producer, broker or consumer is valid for a limited amount of time denoted as *Epoch* and thus does not require explicit key revocation. With the private key of a credential

$$Pr^A = (adv||a_i||A||Epoch)$$

a producer can sign any produced event for which the credential was valid. In particular, let adv denote a dz-string[1] identifying the value range for attribute a_i in which the producer is authorized to publish events. Before publishing an event e, the producer will encrypt the content of its attributes. Let $dz(e)$ denote the dz-string of e then any consumer with a matching subscription needs to be able to decipher e. Since each matching subscription of e can be represented by a prefix of $dz(e)$, a producer will generate for each prefix of $dz(e)$ a

1 a binary string approximating a subspace of the event space, see Chapter 3

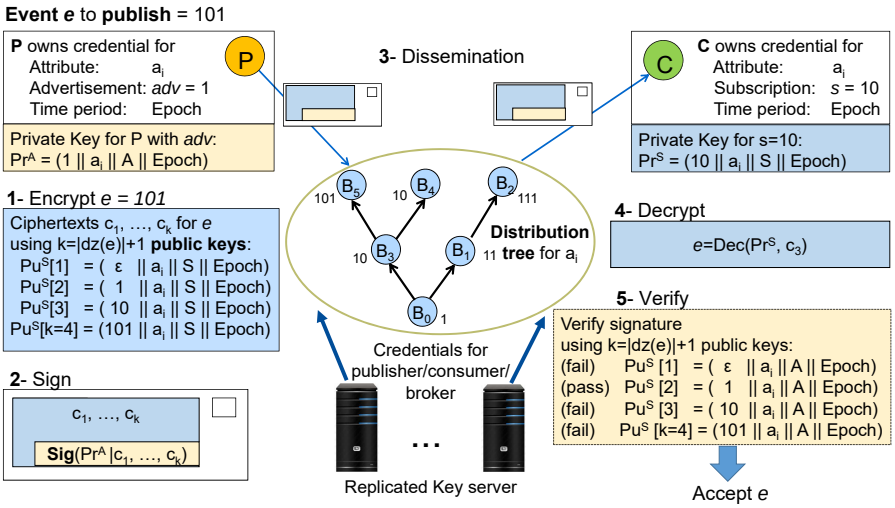


Figure 7.3: An example illustrating how event can be secured using identities corresponding to dz-expressions.

distinct ciphertext. For example, in Figure 7.3 the event "101" will be encoded as four ciphertexts $c_1, \ldots, c_4$ for consumers obtaining a credential with respect to dz-strings in $\{"\epsilon", "1", "10", "101"\}$. Each ciphertext, can be generated with the known public key of the master and the corresponding prefix of $dz(e)$. The process of encrypting events does not require additional key exchanges.

Any broker obtaining one of those credentials can help routing the event in direction to the consumer with subscription "10". Furthermore, the consumer can decrypt the event by applying its credential to the cyphertext c_3. Finally, the consumer can validate the authenticity by verifying the signature against any prefix of e, using the corresponding public key. Since in the example e was signed with respect to the advertisement $adv = "1"$ only the test for the corresponding public key

$$Pu^S[2] = (adv||a_i||A||Epoch)$$

will pass. Note that the producer may explicitly indicate the length of the signature to minimize the number of verification tests. In turn also the producer reveals the size of the subspace in which it is interested to publish events.

Consider the number of advertisements $|A|$ and the number of distinct subscriptions $|S|$ that overlap with any produced event in the worst case. Standard asymmetric cryptography requires $O(|A||S|)$ key exchanges between a producer and consumer. The dz-based key management will reduce the complexity for key exchanges to $O(|A| + |S|)$. Every additional subscription and publication requires only a constant number of keys to be exchanged with the key service. The complexity of the number of cyphertexts and verfication tests for

identity encryption can be bounded by $O(\log(|S|)$ since every cyphertext or verification corresponds to the number of possible prefixes of $dz(e)$.

A further critical issue are suitable mechanisms for performing efficient event routing. Similar to the former findings of Chapter 3 the broker overlay can be completely realized in a decentralized fashion, i. e., realized by producers and consumers. For doing so, the brokers only have to comply to the following invariant (also discussed in Section 3.3): Brokers connect to one broker with same or coarser subscriptions, and accept connections of brokers with the same or finer subscriptions. Events will be routed bottom up in direction to brokers with coarser subscriptions. In addition, a broker which can decrypt an event will also forward the event to all its children except the broker from which the event was received. Similarly, consider a broker B which connects with a subscription to a broker B'. In case B cannot prove that its subscription is more fine-grained than the subscriptions of broker B', B' will redirect B to a broker B'' which is the parent of B'. Otherwise, broker B' may decide, dependent on the number of connected brokers, to accept the connection request of B' or redirect B to one of its children.

As part of this process the brokers can learn something about the interest of parents and children, namely whether they have finer or coarser subscriptions. Therefore, the proposed security method ensures only a weak form of subscription confidentiality.

8

CONCLUSIONS

In this thesis, we highlighted intrinsic principles of building scalable and robust event-based systems. In particular, the thesis highlighted five important aspects of adapting event-based communications:

1. scalable and efficient distribution of events,
2. the elastic parallel execution of event detection,
3. dynamic migration of operators,
4. reliable execution of event processing,
5. and secure access control to events.

For each of the aspects, we identified important problems and challenges, summarized related work, as well as exemplified solutions in a tutorial style building on own research contributions. Understanding the principles of these mechanisms is an important foundation to build large scale distributed applications. For example, robust and scalable big data systems and IoT systems can highly benefit from the adaptation principles in executing operators in a distributed environment in order to meet important application requirements.

In our contributions, we have shown that the spatial indexing model has many interesting properties to build highly scalable event-based systems, execute publish/subscribe in hardware, and account for confidentiality when forwarding events. A second guiding principle underlying this work is accounting for the state of the operator's execution. By proposing a simple execution model of the operator's execution, we have shown that important requirements for the robust execution of event-based systems can be accomplished, by facilitating the parallelization, the migration, and replication of operator's state. Although currently there exists a wide set of languages and flavors of event-based systems, the underlying principles are already influencing the design of many real -world applications.

While at present they have been accomplished forming overlays over IT-infrastructures, in the future event-based systems may be a more and more integral part of IT infrastructures, for example, integrated into future communication networks. This becomes apparent in the area of Information-Centric Networking where important principles of event-based systems are already enforced. This poses a tremendous potential in providing better performance and also better guarantees that are of importance to any application requiring real-time analytics, e. g., any application in the domain of cyber-physical

systems and the IoT. In order to benefit better from the ideas and proposals of the thesis, the increased trend to a softwarization of IT infrastructures is expected to play an important role. For example, building —as discussed in Chapter 3— on accelerating performance of publish/subscribe, is a good example of how the softwarization of IT infrastructures can be used in order to improve the performance.

An important research direction is therefore to provide better programming models of event-based systems that allow to benefit from a heterogeneous infrastructure and select the right mechanisms needed for communication systems. Also, future research may not only work towards improving specific mechanisms for event-based communications, but also at dynamically exchanging the mechanisms dependent on the environmental context at hand. Findings in this direction are researched in the collaborative research center MAKI [4]. However, understanding and verifying the correctness and the QoS properties of such systems is still a major challenge for future research.

BIBLIOGRAPHY

[1] Daniel J Abadi et al. "The Design of the Borealis Stream Processing Engine." In: *Proceedings of 2nd Conference on Innovative Data Systems Research (CIDR '05)*. 2005, pp. 277–289.

[2] Asaf Adi and Opher Etzion. "Amit - the situation manager." In: *The International Journal on Very Large Data Bases (VLDB)*. Springer Nature, 2004, 13.2, pp. 177–203. DOI: 10.1007/s00778-003-0108-y.

[3] Lorenzo Affetti, Alessandro Margara, and Gianpaolo Cugola. "FlowDB: Integrating Stream Processing and Consistent State Management." In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. ACM Press, 2017, pp. 134–145. DOI: 10.1145/3093742.3093929.

[4] Bastian Alt et al. "Transitions: A Protocol-Independent View of the Future Internet." In: *Proceedings of the IEEE*. IEEE, 2019, 107.4, pp. 835–846. DOI: 10.1109/JPROC.2019.2895964.

[5] *Apache Kafka*. URL: https://kafka.apache.org/.

[6] *Apache Storm*. URL: http://storm.apache.org.

[7] Ron Avnur and Joseph M. Hellerstein. "Eddies: continuously adaptive query processing." In: *ACM SIGMOD Record*. ACM Press, 2000, 29.2, pp. 261–272. DOI: 10.1145/335191.335420.

[8] Sébastien Baehni, Patrick T. Eugster, and Rachid Guerraoui. "Data-aware multicast." In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*. IEEE, 2004, 10 pages. DOI: 10.1109/dsn.2004.1311893.

[9] Magdalena Balazinska, Hari Balakrishnan, Samuel R. Madden, and Michael Stonebraker. "Fault-tolerance in the borealis distributed stream processing system." In: *ACM Transactions on Database Systems*. ACM Press, 2008, 33.1, pp. 1–44. DOI: 10.1145/1331904.1331907.

[10] Cagri Balkesen, Nihal Dindar, Matthias Wetter, and Nesime Tatbul. "RIP: run-based intra-query parallelism for scalable complex event processing." In: *Proceedings of the 7th ACM International Conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 3–14. DOI: 10.1145/2488222.2488257.

[11] 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)*. IEEE, 1999. DOI: 10.1109/icdcs.1999.776528.

[12] Roger S. Barga, Jonathan Goldstein, Mohamed Ali, and Mingsheng Hong. "Consistent Streaming Through Time: A Vision for Event Stream Processing." In: *Proceedings of the 3rd Biennial Conference on Innovative Data Systems (CIDR '07)*. 2007, pp. 363–374.

[13] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. "Nephele/PACTs: a programming model and execution framework for web-scale analytical processing." In: *Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10)*. ACM Press, 2010, pp. 119–130. DOI: 10.1145/1807128.1807148.

[14] András Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. "Role-based access control for publish/subscribe middleware architectures." In: *Proceedings of the 2nd international workshop on Distributed event-based systems (DEBS '03)*. ACM Press, 2003, pp. 1–8. DOI: 10.1145/966618.966622.

[15] S. Bhola, R. Strom, S. Bagchi, Yuanyuan Zhao, and J. Auerbach. "Exactly-once delivery in a content-based publish-subscribe system." In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN '02)*. IEEE, 2002, 10 pages. DOI: 10.1109/dsn.2002.1028881.

[16] Sukanya Bhowmik, Muhammad Adnan Tariq, Lobna Hegazy, and Kurt Rothermel. "Hybrid Content-Based Routing Using Network and Application Layer Filtering." In: *Proceedings of the 36th IEEE International Conference on Distributed Computing Systems (ICDCS'16)*. IEEE, 2016, 11 pages. DOI: 10.1109/icdcs.2016.16.

[17] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, Frank Dürr, Thomas Kohler, and Kurt Rothermel. "High Performance Publish/Subscribe Middleware in Software-Defined Networks." In: *IEEE/ACM Transactions on Networking*. IEEE, 2017, 25.3, pp. 1501–1516. DOI: 10.1109/tnet.2016.2632970.

[18] Sukanya Bhowmik, Muhammad Adnan Tariq, Boris Koldehofe, André Kutzleb, and Kurt Rothermel. "Distributed control plane for software-defined networks." In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based*

Systems (DEBS '15). ACM Press, 2015. DOI: 10.1145/2675743.2771835.

[19] Irina Botan, Peter M. Fischer, Donald Kossmann, and Nesime Tatbul. "Transactional stream processing." In: *Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12)*. ACM Press, 2012, pp. 204–215. DOI: 10.1145/2247596.2247622.

[20] Jorge A. Briones, Boris Koldehofe, and Kurt Rothermel. "SPINE: Adaptive Publish/Subscribe for Wireless Mesh Networks." In: *Studia Informatika Universalis*. Hermann, 2009, 7.3, pp. 320–353.

[21] Andrey Brito, Christof Fetzer, Heiko Sturzrehm, and Pascal Felber. "Speculative out-of-order event processing with software transaction memory." In: *Proceedings of the 2nd International Conference on Distributed Event-based Systems (DEBS '08)*. ACM Press, 2008, pp. 265–275. DOI: 10.1145/1385989.1386023.

[22] Andrey Brito, Andre Martin, Thomas Knauth, Stephan Creutz, Diogo Becker, Stefan Weigert, and Christof Fetzer. "Scalable and Low-Latency Data Processing with Stream MapReduce." In: *Proceedings of the 3rd International IEEE Conference on Cloud Computing Technology and Science (CloudCom '08)*. IEEE, 2011, pp. 48–58. DOI: 10.1109/cloudcom.2011.17.

[23] Alejandro Buchmann and Boris Koldehofe. "Complex Event Processing." In: *it - Information Technology*. de Gruyter, 2009, 51.5, pp. 241–242. DOI: 10.1524/itit.2009.9058.

[24] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. "Design and evaluation of a wide-area event notification service." In: *ACM Transactions on Computer Systems*. ACM Press, 2001, 19.3, pp. 332–383. DOI: 10.1145/380749.380767.

[25] S. Chakravarthy and D. Mishra. "Snoop: An expressive event specification language for active databases." In: *Data & Knowledge Engineering*. Elsevier, 1994, 14.1, pp. 1–26. DOI: 10.1016/0169-023x(94)90006-x.

[26] Tushar Deepak Chandra and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." In: *Journal of the ACM*. ACM Press, 1996, 43.2, pp. 225–267. DOI: 10.1145/226643.226647.

[27] Sunoh Choi, Gabriel Ghinita, and Elisa Bertino. "A Privacy-Enhancing Content-Based Publish/Subscribe System Using Scalar Product Preserving Transformations." In: *Proceedings of the Database and Expert Systems Applications (DEXA '10)*. Springer, 2010, pp. 368–384. DOI: 10.1007/978-3-642-15364-8_32.

[28] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. "MapReduce online." In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI '10)*. USENIX Association, 2010.

[29] G. Cugola, E. Di Nitto, and A. Fuggetta. "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS." In: *IEEE Transactions on Software Engineering*. IEEE, 2001, 27.9, pp. 827–850. DOI: 10.1109/32.950318.

[30] Gianpaolo Cugola and Alessandro Margara. "TESLA: a formally defined event specification language." In: *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS '10)*. ACM Press, 2010, pp. 50–61. DOI: 10.1145/1827418.1827427.

[31] Gianpaolo Cugola and Alessandro Margara. "Deployment Strategies for Distributed Complex Event Processing." English. In: *Springer Computing*. Springer, 2013, 95.2, pp. 129–156. DOI: 10.1007/s00607-012-0217-9.

[32] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. "Vivaldi: a decentralized network coordinate system." In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*. ACM Press, 2004, pp. 15–26. DOI: 10.1145/1015467.1015471.

[33] Nihal Dindar, Peter M. Fischer, Merve Soner, and Nesime Tatbul. "Efficiently correlating complex events over live and archived data streams." In: *Proceedings of the 5th ACM international conference on Distributed Event-Based Systems (DEBS '11)*. ACM Press, 2011. DOI: 10.1145/2002259.2002293.

[34] D. Dolev and H. R. Strong. "Authenticated Algorithms for Byzantine Agreement." In: *SIAM Journal on Computing*. SIAM, 1983, 12.4, pp. 656–666. DOI: 10.1137/0212045.

[35] Rahul Dwarakanath, Boris Koldehofe, Yashas Bharadwaj, The An Binh Nguyen, David Eyers, and Ralf Steinmetz. "TrustCEP: Adopting a Trust-Based Approach for Distributed Complex Event Processing." In: *Proceedings of the 18th IEEE International Conference on Mobile Data Management (MDM '17)*. IEEE, 2017, pp. 30–39. DOI: 10.1109/mdm.2017.15.

[36] Rahul Dwarakanath, Boris Koldehofe, and Ralf Steinmetz. "Operator Migration for Distributed Complex Event Processing in Device-to-Device Based Networks." In: *Proceedings of the 3rd Workshop on Middleware for Context-Aware Applications in the IoT (M4IoT '16)*. ACM Press, 2016, pp. 13–18. DOI: 10.1145/3008631.3008634.

[37] Kyumars Sheykh Esmaili, Tahmineh Sanamrad, Peter M. Fischer, and Nesime Tatbul. "Changing flights in mid-air." In: *Proceedings of the International Conference on Management of Data (SIGMOD '11)*. ACM Press, 2011. DOI: 10.1145/1989323.1989388.

[38] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The many faces of publish/subscribe." In: *ACM Computing Surveys*. ACM Press, 2003, 35.2, pp. 114–131. DOI: 10.1145/857076.857078.

[39] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. "Integrating scale out and fault tolerance in stream processing using operator state management." In: *Proceedings of the International Conference on Management of Data (SIGMOD '13)*. ACM Press, 2013, pp. 725–736. DOI: 10.1145/2463676.2465282.

[40] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." In: *Journal of the ACM*. ACM Press, 1985, 32.2, pp. 374–382. DOI: 10.1145/3149.214121.

[41] Stella Gatziu and Klaus R. Dittrich. "SAMOS: An active object-oriented database system." In: *IEEE Data Engineering Bulletin*. IEEE, 1992, 15.1-4, pp. 23–26.

[42] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. "SPADE: the system s declarative stream processing engine." In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data (SIGMOD '08)*. ACM Press, 2008, pp. 1123–1134. DOI: 10.1145/1376616.1376729.

[43] Bugra Gedik and Ling Liu. "MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries." In: *IEEE Transactions on Mobile Computing*. IEEE, 2006, 5, pp. 1384–1402. DOI: 10.1109/TMC.2006.153.

[44] Yu Gu, Ge Yu, Na Guo, and Yueguo Chen. "Probabilistic Moving Range Query over RFID Spatio-temporal Data Streams." In: *Proceedings of the 18th ACM conference on Information and knowledge management (CIKM '09)*. 2009, pp. 1413–1416. DOI: 10.1145/1645953.1646133.

[45] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. "StreamCloud: An Elastic and Scalable Data Streaming System." In: *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2012, 23.12, pp. 2351–2365. DOI: 10.1109/tpds.2012.24.

[46] Thomas Heinze, Mariam Zia, Robert Krahn, Zbigniew Jerzak, and Christof Fetzer. "An adaptive replication scheme for elastic data stream processing systems." In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS '15)*. ACM Press, 2015, pp. 150–161. DOI: 10.1145/2675743.2771831.

[47] Kirak Hong, Stephen Smaldone, Junsuk Shin, David Lillethun, Liviu Iftode, and Umakishore Ramachandran. "Target container: A target-centric parallel programming abstraction for video-based surveillance." In: *Proceedings of the 5th ACM/IEEE International Conference on Distributed Smart Cameras (ICDCSC '11)*. IEEE, 2011, 8 pages. DOI: 10.1109/icdsc.2011.6042914.

[48] Waldemar Hummer, Philipp Leitner, Benjamin Satzger, and Schahram Dustdar. "Dynamic Migration of Processing Elements for Optimized Query Execution in Event-based Systems." In: *Proceedings of the On the Move to Meaningful Internet Systems (OTM '11)*. OTM '11. Springer-Verlag, 2011, pp. 451–468. DOI: 10.1007/978-3-642-25106-1_3.

[49] Jeong-Hyon Hwang, Magdalena Balazinska, Alexander Rasin, Uğur Çetintemel, Michael Stonebraker, and Stan Zdonik. "High-Availability Algorithms for Distributed Stream Processing." In: *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. IEEE, 2005. DOI: 10.1109/icde.2005.72.

[50] Jeong-Hyon Hwang, Ugur Cetintemel, and Stan Zdonik. "Fast and Highly-Available Stream Processing over Wide Area Networks." In: *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE '08)*. IEEE, 2008, pp. 804–813. DOI: 10.1109/icde.2008.4497489.

[51] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. "Dryad: distributed data-parallel programs from sequential building blocks." In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*. ACM Press, 2007, pp. 59–72. DOI: 10.1145/1272996.1273005.

[52] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. "The PADRES Publish/Subscribe System." In: *Principles and Applications of Distributed Event-Based Systems*. IGI Global, pp. 164–205. DOI: 10.4018/978-1-60566-697-6.ch008.

[53] Hans-Arno Jacobsen, Vinod Muthusamy, and Guoli Li. "The PADRES Event Processing Network: Uniform Querying of Past and Future EventsDas PADRES." In: *it - Information Tech-*

nology. de Gruyter, 2009, 51.5, pp. 250–261. DOI: 10.1524/itit.2009.0549.

[54] Gabriela Jacques-Silva, Bugra Gedik, Henrique Andrade, and Kun-Lung Wu. "Language level checkpointing support for stream processing applications." In: *Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks (DSN '09)*. IEEE, 2009, pp. 145–154. DOI: 10.1109/dsn.2009.5270344.

[55] K. Jayaram, Chamikara Jayalath, and Patrick Eugster. "Parametric Subscriptions for Content-Based Publish/Subscribe Networks." In: *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware (Middleware '10)*. 2010, pp. 128–147. DOI: 10.1007/978-3-642-16955-7_7.

[56] Xing Jin, Wanqing Tu, and S.-H.G. Chan. "Scalable and Efficient End-to-End Network Topology Inference." In: *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2008, 19.6, pp. 837–850. DOI: 10.1109/tpds.2007.70771.

[57] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. "LIPSIN: line speed publish/subscribe inter-networking." In: *Proceedings of the ACM SIGCOMM Conference on Data communication (SIGCOMM '09)*. ACM Press, 2009, pp. 195–206. DOI: 10.1145/1592568.1592592.

[58] Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel. "Cordies: expressive event correlation in distributed systems." In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS '10)*. ACM Press, 2010, pp. 26–37. DOI: 10.1145/1827418.1827424.

[59] Thomas Kohler, Ruben Mayer, Frank Dürr, Marius Maaß, Sukanya Bhowmik, and Kurt Rothermel. "P4CEP: Towards In-Network Complex Event Processing." In: *Proceedings of the Morning Workshop on In-Network Computing (NetCompute '18)*. ACM Press, 2018, pp. 33–38. DOI: 10.1145/3229591.3229593.

[60] Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, and Marco Völz. "Rollback-recovery without checkpoints in distributed event processing systems." In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 27–38. DOI: 10.1145/2488222.2488259.

[61] Boris Koldehofe, Beate Ottenwälder, Kurt Rothermel, and Umakishore Ramachandran. "Moving range queries in distributed complex event processing." In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*

(DEBS '12). ACM Press, 2012, pp. 201–212. DOI: 10.1145/2335484.2335507.

[62] R. Koo and S. Toueg. "Checkpointing and Rollback-Recovery for Distributed Systems." In: *IEEE Transactions on Software Engineering*. IEEE, 1987, SE-13.1, pp. 23–31. DOI: 10.1109/tse.1987.232562.

[63] Minseok Kwon and Sonia Fahmy. "Path-aware overlay multicast." In: *Computer Networks*. Elsevier, 2005, 47.1, pp. 23–45. DOI: 10.1016/s1389-1286(04)00202-6.

[64] YongChul Kwon, Magdalena Balazinska, and Albert Greenberg. "Fault-tolerant stream processing using a distributed, replicated file system." In: *Proceedings of the VLDB Endowment*. VLDB Endowment, 2008, 1.1, pp. 574–585. DOI: 10.14778/1453856.1453920.

[65] Geetika T. Lakshmanan, Ying Li, and Rob Strom. "Placement Strategies for Internet-Scale Data Stream Systems." In: *IEEE Internet Computing*. Piscataway, NJ, USA: IEEE, Nov. 2008, 12.6, pp. 50–60. ISSN: 1089-7801. DOI: 10.1109/MIC.2008.129.

[66] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system." In: *Communications of the ACM*. ACM Press, 1978, 21.7, pp. 558–565. DOI: 10.1145/359545.359563.

[67] Ming Li, Mo Liu, Luping Ding, Elke A. Rundensteiner, and Murali Mani. "Event Stream Processing with Out-of-Order Data Arrival." In: *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCSW '07)*. IEEE, 2007, 8 pages. DOI: 10.1109/icdcsw.2007.35.

[68] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison Wesley, 2002. ISBN: 978-0201727890.

[69] Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. "Decentralized Message Ordering for Publish/Subscribe Systems." In: *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware (Middleware'07)*. Springer, 2006, pp. 162–179. DOI: 10.1007/11925071_9.

[70] Manisha Luthra, Boris Koldehofe, Pascal Weisenburger, Guido Salvaneschi, and Raheel Arif. "TCEP: Adapting to Dynamic User Environments by Enabling Transitions between Operator Placement Mechanisms." In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18)*. ACM Press, 2018. DOI: 10.1145/3210284.3210292.

[71] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. "Predictable Low-Latency Event Detection With Parallel Complex Event Processing." In: *IEEE Internet of Things Journal*. IEEE, 2015, 2.4, pp. 274–286. DOI: 10.1109/jiot.2015.2397316.

[72] Ruben Mayer, Ahmad Slo, Muhammad Adnan Tariq, Kurt Rothermel, Manuel Gräber, and Umakishore Ramachandran. "SPECTRE: supporting consumption policies in window-based parallel complex event processing." In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference on (Middleware '17*. ACM Press, 2017, pp. 161–173. DOI: 10.1145/3135974.3135983.

[73] Sarah McCarthy, Neil Smyth, and Elizabeth O'Sullivan. "A Practical Implementation of Identity-Based Encryption Over NTRU Lattices." In: *Cryptography and Coding*. Springer International Publishing, 2017, pp. 227–246. DOI: 10.1007/978-3-319-71045-7_12.

[74] Gero Mühl. "Large-Scale Content-Based Publish-Subscribe Systems." PhD thesis. TU Darmstadt, 2002.

[75] Javier Munster and Hans-Arno Jacobsen. "Secret Sharing in Pub/Sub Using Trusted Execution Environments." In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18)*. ACM Press, 2018, pp. 28–39. DOI: 10.1145/3210284.3210290.

[76] Christopher Mutschler and Michael Philippsen. "Adaptive Speculative Processing of Out-of-Order Event Streams." In: *ACM Transactions on Internet Technology*. ACM Press, 2014, 14.1, pp. 1–24. DOI: 10.1145/2633686.

[77] Mohamed Nabeel, Ning Shang, and Elisa Bertino. "Efficient privacy preserving content based publish subscribe systems." In: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies (SACMAT '2012)*. ACM Press, 2012, pp. 133–144. DOI: 10.1145/2295136.2295164.

[78] Dan O'Keeffe and Jean Bacon. "Reliable complex event detection for pervasive computing." In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems - DEBS '10*. ACM Press, 2010, pp. 73–84. DOI: 10.1145/1827418.1827429.

[79] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. "Confidentiality-Preserving Publish/Subscribe." In: *ACM Computing Surveys*. ACM Press, 2016, 49.2, pp. 1–43. DOI: 10.1145/2940296.

[80] Lukasz Opyrchal and Atul Prakash. "Secure distribution of events in content-based publish subscribe systems." In: *Proceedings of the 10th conference on USENIX Security Symposium (SSYM '01)*. 2001, pp. 281–296.

[81] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, David Lillethun, and Umakishore Ramachandran. "MCEP: A Mobility-Aware Complex Event Processing System." In: *ACM Transactions on Internet Technology*. ACM Press, 2014, 14.1, pp. 1–24. DOI: 10.1145/2633688.

[82] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, Kirak Hong, and Umakishore Ramachandran. "RECEP: selection-based reuse for distributed complex event processing." In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS '14)*. ACM Press, 2014, pp. 59–70. DOI: 10.1145/2611286.2611297.

[83] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. "MigCEP: operator migration for mobility driven distributed complex event processing." In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 183–194. DOI: 10.1145/2488222.2488265.

[84] Saravana Murthy Palanisamy, Frank Dürr, Muhammad Adnan Tariq, and Kurt Rothermel. "Preserving Privacy and Quality of Service in Complex Event Processing through Event Reordering." In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18)*. ACM Press, 2018, pp. 40–51. DOI: 10.1145/3210284.3210296.

[85] Lauri I. W. Pesonen, David M. Eyers, and Jean Bacon. "Encryption-enforced access control in dynamic multi-domain publish/subscribe networks." In: *Proceedings of the 2007 Inaugural International Conference on Distributed event-based systems (DEBS '07)*. ACM Press, 2007, pp. 104–115. DOI: 10.1145/1266894.1266916.

[86] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. "Network-Aware Operator Placement for Stream-Processing Systems." In: *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*. IEEE, 2006, 12 pages. DOI: 10.1109/icde.2006.105.

[87] Peter Pietzuch. "Hermes: A Scalable Event-Based Middleware." PhD thesis. University of Cambridge, 2004.

[88] Costin Raiciu and David S. Rosenblum. "Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures." In: *Proceedings of the 2006 Securecomm and Workshops*. IEEE, 2006, 11 pages. DOI: 10.1109/seccomw.2006.359552.

[89] B. Randell, P. Lee, and P. C. Treleaven. "Reliability Issues in Computing System Design." In: *ACM Computing Surveys*. ACM Press, 1978, 10.2, pp. 123–165. DOI: 10.1145/356725.356729.

[90] Björn Richerzhagen, Nils Richerzhagen, Julian Zobel, Sophie Schönherr, Boris Koldehofe, and Ralf Steinmetz. "Seamless Transitions between Filter Schemes for Location-Based Mobile Applications." In: *Proceedings of the 41st IEEE Conference on Local Computer Networks (LCN '16)*. IEEE, 2016, pp. 348–356. DOI: 10.1109/lcn.2016.28.

[91] Stamatia Rizou, Frank Durr, and Kurt Rothermel. "Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks." In: *Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN '10)*. IEEE, 2010, 6 pages. DOI: 10.1109/icccn.2010.5560127.

[92] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. "SCRIBE: The Design of a Large-Scale Event Notification Infrastructure." In: *Proceedings of the Third International COST264 Workshop on Networked Group Communication (NGC '01)*. Springer., 2001, pp. 30–43. DOI: 10.1007/3-540-45546-9_3.

[93] Mohammad Sadoghi, Harsh Singh, and Hans-Arno Jacobsen. "fpga-ToPSS: line-speed event processing on fpgas." In: *Proceedings of the 5th ACM International Conference on Distributed Event-based Systems (DEBS '11)*. ACM Press, 2011, pp. 373–374. DOI: 10.1145/2002259.2002316.

[94] Björn Schilling, Boris Koldehofe, and Kurt Rothermel. "Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems." In: *Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC '11)*. IEEE, 2011, pp. 355–364. DOI: 10.1109/hpcc.2011.53.

[95] Björn Schilling, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. "Access Policy Consolidation for Event Processing Systems." In: *Proceedings of the Conference on Networked Systems (NetSys '13)*. IEEE, 2013, pp. 92–101. DOI: 10.1109/netsys.2013.18.

[96] Fred B. Schneider. "Implementing fault-tolerant services using the state machine approach: a tutorial." In: *ACM Computing Surveys*. ACM Press, 1990, 22.4, pp. 299–319. DOI: 10.1145/98163.98167.

[97] Scott Schneider, Henrique Andrade, Bugra Gedik, Alain Biem, and Kun-Lung Wu. "Elastic scaling of data parallel operators in stream processing." In: *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDS '09)*. IEEE, 2009, 12 pages. DOI: 10.1109/ipdps.2009.5161036.

[98] Scott Schneider, Martin Hirzel, Bugra Gedik, and Kun-Lung Wu. "Auto-parallelizing stateful distributed streaming applications." In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*. IEEE, 2012, pp. 53–63.

[99] Reinhard Schwarz and Friedemann Mattern. "Detecting causal relationships in distributed computations: In search of the holy grail." In: *Distributed Computing*. Springer Nature, 1994, 7.3, pp. 149–174. DOI: 10.1007/bf02277859.

[100] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes." In: *Advances in Cryptology*. Springer, pp. 47–53. DOI: 10.1007/3-540-39568-7_5.

[101] Josef Spillner, Martin Beck, Alexander Schill, and Thomas Michael Bohnert. "Stealth Databases: Ensuring User-Controlled Queries in Untrusted Cloud Environments." In: *Proceedings of the 8th International Conference on Utility and Cloud Computing (UCC '15)*. 2015, pp. 261–270.

[102] Mudhakar Srivatsa and Ling Liu. "Secure Event Dissemination in Publish-Subscribe Networks." In: *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS '07)*. IEEE, 2007, 8 pages. DOI: 10.1109/icdcs.2007.136.

[103] Mudhakar Srivatsa, Ling Liu, and Arun Iyengar. "EventGuard." In: *ACM Transactions on Computer Systems*. ACM Press, 2011, 29.4, pp. 1–40. DOI: 10.1145/2063509.2063510.

[104] Fabrice Starks, Vera Goebel, Stein Kristiansen, and Thomas Plagemann. "Mobile Distributed Complex Event Processing—Ubi Sumus? Quo Vadimus?" In: *Mobile Big Data*. Springer International Publishing, 2017, pp. 147–180. DOI: 10.1007/978-3-319-67925-9_7.

[105] Jimeng Sun, D. Papadias, Yufei Tao, and Bin Liu. "Querying about the Past, the Present, and the Future in Spatio-Temporal Databases." In: *Proceedings of the 20th International Conference on Data Engineering (ICDE '04)*. ICDE '04. 2004, pp. 202–213. DOI: 10.1109/ICDE.2004.1319997.

[106] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, and Kurt Rothermel. "Distributed spectral cluster management: a method for building dynamic publish/subscribe systems." In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12)*. ACM Press, 2012, pp. 213–224. DOI: 10.1145/2335484.2335508.

[107] Muhammad Adnan Tariq, Boris Koldehofe, Gerald Georg Koch, Imran Khan, and Kurt Rothermel. "Meeting subscriber-defined QoS constraints in publish/subscribe systems." In: *Concurrency and Computation: Practice and Experience*. Wiley, 2011, 23.11, pp. 2140–2153. DOI: 10.1002/cpe.1751.

[108] Muhammad Adnan Tariq, Boris Koldehofe, Gerald Koch, and Kurt Rothermel. "Providing Probabilistic Latency Bounds for Dynamic Publish/Subscribe Systems." In: *Proceedings of the 16th ITG/GI Conference on Kommunikation in Verteilten Systemen (KiVS '09)*. 2009, pp. 155–166. DOI: 10.1007/978-3-540-92666-5_13.

[109] Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. "Efficient content-based routing with network topology inference." In: *Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13)*. ACM Press, 2013, pp. 51–62. DOI: 10.1145/2488222.2488262.

[110] Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. "Securing Broker-Less Publish/Subscribe Systems Using Identity-Based Encryption." In: *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 2014, 25.2, pp. 518–528. DOI: 10.1109/tpds.2013.256.

[111] Marco Völz, Boris Koldehofe, and Kurt Rothermel. "Supporting Strong Reliability for Distributed Complex Event Processing Systems." In: *Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC '11)*. IEEE, 2011, pp. 477–486. DOI: 10.1109/hpcc.2011.69.

[112] Oliver P. Waldhorst, Christian Blankenhorn, Dirk Haage, Ralph Holz, Gerald G Koch, Boris Koldehofe, Fleming Lampi, Christoph P. Mayer, and Sebastian Mies. "Spontaneous Virtual Networks: On the Road Towards the Internet´s Next Generation." In: *it - Information Technology*. de Gruyter, 2009, 50.6, pp. 367–375. DOI: 10.1524/itit.2008.0508.

[113] Chenxi Wang, A. Carzaniga, D. Evans, and A.L. Wolf. "Security issues and requirements for Internet-scale publish-subscribe systems." In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS '02)*. IEEE, 2002, 8 pages. DOI: 10.1109/hicss.2002.994531.

[114] Pascal Weisenburger, Manisha Luthra, Boris Koldehofe, and Guido Salvaneschi. "Quality-Aware Runtime Adaptation in Complex Event Processing." In: *Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '17)*. IEEE, 2017. DOI: 10.1109/seams.2017.10.

[115] Eugene Wu, Yanlei Diao, and Shariq Rizvi. "High-performance complex event processing over streams." In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*. ACM Press, 2006, pp. 407–418. DOI: 10.1145/1142473.1142520.

[116] Xiaopeng Xiong, H.G. Elmongui, Xiaoyong Chai, and Walid G. Aref. "PLACE*: A Distributed Spatio-temporal Data Stream Management System for Moving Objects." In: *Proceedings of the International Conference on Mobile Data Management (MDM '18)*. MDM '07. 2007, pp. 44–51. DOI: 10.1109/MDM.2007.16.

[117] Kaiwen Zhang, Vinod Muthusamy, and Hans-Arno Jacobsen. "Total Order in Content-Based Publish/Subscribe Systems." In: *Proceedings of the IEEE 32nd International Conference on Distributed Computing Systems (ICDCS '12)*. IEEE, 2012. DOI: 10.1109/icdcs.2012.17.