


3-23-2015

Temporal Mining for Distributed Systems

Yexi Jiang

School of Computing and Information Sciences, yjian004@fiu.edu

Follow this and additional works at: <http://digitalcommons.fiu.edu/etd>

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Recommended Citation

Jiang, Yexi, "Temporal Mining for Distributed Systems" (2015). *FIU Electronic Theses and Dissertations*. Paper 1909.
<http://digitalcommons.fiu.edu/etd/1909>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

TEMPORAL MINING FOR DISTRIBUTED SYSTEMS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Yexi Jiang

2015

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Yexi Jiang, and entitled Temporal Mining for Distributed Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Sundaraja Sitharama Iyengar

Shu-Ching Chen

Jinpeng Wei

Wensong Wu

Tao Li, Major Professor

Date of Defense: March 23, 2015

The dissertation of Yexi Jiang is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2015

© Copyright 2015 by Yexi Jiang

All rights reserved.

DEDICATION

To my wife and family.

ACKNOWLEDGMENTS

After three degrees, at two universities, I learnt that I could never have done any of this, particularly the research and writing that went into this dissertation, without the kind help and encouragement of the people who support me.

First, I would like to thank my advisor, Professor Tao Li, who has given so much to help me succeed. Thank you for teaching me a lot of useful things during my Ph.D. career, with endless patience.

I would also thank all my thesis committee members: Professor Sundaraja Sitharama Iyengar, Professor Shu-Ching Chen, Professor Jinpeng Wei, and Professor Wensong Wu. Thank you for your helpful advices and insightful comments on my research, my thesis, and my future research career plans.

Moreover, I'd like to thank all my mentors for my internships at IBM and Facebook: Dr. Charles Perng, Dr. Chang Rong, Dr. Anca Sailer, Dr. Ignacio Silva-Lepe, Dr. David Vickrey, and Dr. Jie Xu. Thank you all for your patient guidance and help during my internships. Those summers are unforgettable.

I've also been fortunate to have a great group of labmates in FIU: Dr. Jingxuan Li, Dr. Lei Li, Dr. Li Zhen, Dr. Dingding Wang, Dr. Yi Zhang, Dr. Liang Tang, Dr. Chao Shen, Yahya Benhadda, Chunqiu Zeng, Longhui Zhang, Hongtai Li, Wubai Zhou, and Wei Xue. Not only are you the people I can discuss my research, but also you are confidants who I can discuss my troubles with and who stand by me through thick and thin.

Finally, I would like to dedicate this work to my wife Jing Liu. Without your unending support, understanding, and love, I never would have made it through this process or any of the tough times in my life. Thank you.

ABSTRACT OF THE DISSERTATION
TEMPORAL MINING FOR DISTRIBUTED SYSTEMS

by

Yexi Jiang

Florida International University, 2015

Miami, Florida

Professor Tao Li, Major Professor

Many systems and applications are continuously producing events. These events are used to record the status of the system and trace the behaviors of the systems. By examining these events, system administrators can check the potential problems of these systems. If the temporal dynamics of the systems are further investigated, the underlying patterns can be discovered. The uncovered knowledge can be leveraged to predict the future system behaviors or to mitigate the potential risks of the systems. Moreover, the system administrators can utilize the temporal patterns to set up event management rules to make the system more intelligent.

With the popularity of data mining techniques in recent years, these events gradually become more and more useful. Despite the recent advances of the data mining techniques, the application to system event mining is still in a rudimentary stage. Most of works are still focusing on episodes mining or frequent pattern discovering. These methods are unable to provide a brief yet comprehensible summary to reveal the valuable information from the high level perspective. Moreover, these methods provide little actionable knowledge to help the system administrators to better manage the systems. To better make use of the recorded events, more practical techniques are required.

From the perspective of data mining, three correlated directions are considered to be helpful for system management: (1) Provide concise yet comprehensive summaries about the running status of the systems; (2) Make the systems more intelligence and

autonomous; (3) Effectively detect the abnormal behaviors of the systems. Due to the richness of the event logs, all these directions can be solved in the data-driven manner. And in this way, the robustness of the systems can be enhanced and the goal of autonomous management can be approached.

This dissertation mainly focuses on the foregoing directions that leverage temporal mining techniques to facilitate system management. More specifically, three concrete topics will be discussed, including event, resource demand prediction, and streaming anomaly detection. Besides the theoretic contributions, the experimental evaluation will also be presented to demonstrate the effectiveness and efficacy of the corresponding solutions.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	1
1.1 Background	1
1.2 Contribution	6
1.2.1 Effective Algorithm and Framework for Event Summarization	7
1.2.2 A Comprehensive Framework for Making the Cloud More Intelligent	8
1.2.3 Online Anomaly Detection Solution on Distributed Systems	9
1.3 Chapter Summary	9
2. Related Work	11
2.1 Related Work of Event Summarization	11
2.1.1 Event Summarization Algorithms	11
2.1.2 Minimum Description Length Applications in Data Mining	12
2.1.3 Wavelet Transformation Applications in Data Mining	13
2.2 Related Work of Cloud Demand Prediction	13
2.2.1 System Oriented Data Mining	13
2.2.2 System Behavior Modeling and Prediction	14
2.2.3 Virtual Environment Management	15
2.3 Related Works of Stream Anomaly Detection	15
3. Temporal Mining for Event Summarization	18
3.1 Motivation	18
3.1.1 The Need for A More Intuitive Event Summarization	19
3.1.2 The Need for a General Event Summarization Framework	21
3.2 Natural Event Summarization	23
3.2.1 Approach Overview	23
3.2.2 Definitions and Problem Formulation	25
3.2.3 Summarization using MDL	28
3.2.4 The NES Algorithm	31
3.2.5 Efficiency Improvement	34
3.2.6 Experimental Evaluation	38
3.3 Multi-resolution Event Summarization Framework	47
3.3.1 The Multi-Resolution Data Model	48
3.3.2 Basic Operations	55
3.3.3 Event Summarization Tasks	59
3.3.4 Experimental Evaluation	62
3.4 Chapter Summary	71
4. Temporal Mining for Cloud Demand Prediction	74
4.1 Motivation and Challenges	74
4.1.1 Challenges for Cloud Capacity Management	75
4.1.2 Challenges for On-demand Virtual Machine Provisioning	76

4.2	Problem Formulation	78
4.3	Cloud Demand Prediction System Framework	79
4.4	Cloud Demand Prediction Approach	83
4.4.1	Prediction Result Evaluation Criteria	83
4.4.2	Ensemble Prediction Model	86
4.4.3	Cloud Capacity Prediction	88
4.4.4	Virtual Machine Type-aware Provisioning Prediction	92
4.5	Experimental Evaluation	95
4.5.1	Experiment Setup and Pre-processing	95
4.5.2	Provisioning Prediction Evaluation	100
4.5.3	De-provisioning Prediction Evaluation	102
4.5.4	Type-aware Prediction Evaluation	105
4.5.5	Comparison of Different Measures	111
4.5.6	Model Computational Cost	112
4.6	Chapter Summary	113
5.	Temporal Mining for Stream Anomaly Detection	114
5.1	Background and Motivation	114
5.1.1	A Motivating Example	114
5.2	Problem Statement	117
5.3	Framework Overview	119
5.4	Stream Anomaly Detection Methodology	122
5.4.1	Data Receiving and Dispatching	122
5.4.2	Snapshot Anomaly Quantification	123
5.4.3	Stream Anomaly Quantification	125
5.4.4	Alert Triggering	129
5.5	Experimental Evaluation	131
5.5.1	Real World Scenario — Anomaly Detection of Computing Cluster	132
5.5.2	Real Word Scenario 2 — Twitter Topics Anomaly Detection	138
5.6	Chapter Summary	140
6.	Conclusion	141
	BIBLIOGRAPHY	144
	VITA	162

LIST OF FIGURES

FIGURE	PAGE
3.1 A motivating example for event summarization	20
3.2 Output summary of the proposed approach.	23
3.3 The framework of natural event summarization	25
3.4 (a) Inter-arrival histogram $h_{aa}(S)$; (b) Inter-arrival histogram $h_{ab}(S)$. . .	27
3.5 (a) Standard histogram that best approximate $h_{aa}(S)$; (b) Standard histogram that best approximate $h_{ab}(S)$	28
3.6 An example histogram graph	32
3.7 An example ERN graph.	34
3.8 Segments information in wavelet spectrum sequence.	37
3.9 Before pruning, the histogram graph contains 17 vertices and 136 edges, after pruning, the histogram graph contains only 6 vertices and 14 edges.	37
3.10 CR vs. noise, with parameters	41
3.11 CR vs. proportion of pattern	42
3.12 Scalability of NES and NES-Prune	43
3.13 Running time vs. number of event type	44
3.14 Running time vs. Proportion of Patterns	45
3.15 ERN for security log	46
3.16 Event summarization scenarios and corresponding workflows	48
3.17 Convert the original event sequence to the vectors	49
3.18 Relationship between vector and ST	52
3.19 Compression ratio of SF and compressed SF	64
3.20 Compression ratio of SF in different resolution	65
3.21 Datasets with different #events	68
3.22 Datasets with different #types	69
3.23 Datasets with different #ts	70
3.24 Updating task	71

3.25	Merging task	72
3.26	Summarize with META	72
4.1	The dynamics of customers	81
4.2	Time series of resource demands grouped by customer. The three time series show the demand of three frequent requesting customers. (The scales of all the time series are not normalized for ease of visualization.)	81
4.3	Time series of resource demands grouped by VM types. The three time series show the demand of three frequent requested VM types. (The scales of all the time series are not normalized for ease of visualization. For confidential issue, the value on y-axis are removed.)	82
4.4	Cloud provisioning prediction system framework	82
4.5	Original capacity time series	88
4.6	Capacity change time series	89
4.7	Provisioning/de-provisioning time series	89
4.8	Total requests time series and individual VM type requests time series	94
4.9	Time series with different granularities. From top to bottom, the time series are aggregated by week, day, and hour, respectively.	96
4.10	Request distribution in time-type-request view	97
4.11	CDF of VM types	98
4.12	Frequency order of VM types	99
4.13	The effect of tuning β for capacity prediction	101
4.14	Prediction result of de-provisioning time series	102
4.15	Errors of different methods (normalized)	104
4.16	Prediction result of Red Hat Enterprise Linux 5.5 (32-bit)	106
4.17	Prediction result of Red Hat Enterprise Linux 5.5 (64-bit)	107
4.18	Prediction result SUSE Linux Enterprise Server	108
4.19	Average provisioning time reduction	109
4.20	Average resource waste	110
4.21	The influence of R_{fix} on prediction (normalized data)	111
5.1	CPU utilization of a computing cluster	115

5.2	Identified anomalies in Example 5 (The box lists the IDs of abnormal streams during specified time period)	116
5.3	Distributed real time stream anomaly detection framework	119
5.4	The snapshot at a certain timestamp	120
5.5	Random partition of data instance	120
5.6	Transient fluctuation and anomaly	127
5.7	Abnormal streams identification	129
5.8	Injections and the captured alerts	133
5.9	F-measure versus reset threshold	135
5.10	Time delay versus reset threshold	135
5.11	Generated alerts with the worst recall	136
5.12	Generated alerts by CAD and Rule-CQ	137
5.13	Country dataset	139
5.14	Election dataset	139

Introduction

1.1 Background

Many systems and applications are continuously generating temporal events (sequences of events with associated timestamps), from events such as low level resource utilization trace events, devices operation events, OS operation events, HTTP request events, and network traffic events, to high level events such as database queries events, UI click events, user behavior events, and virtual machine demand request events. These recorded events are used to capture the system status and to trace the behaviors of the systems over time. By examining these events, system administrators can check the potential problems of the systems. If the temporal dynamics of these events are further investigated, the discovered temporal patterns can be leveraged to predict the future behavior or to mitigate the potential risks of the systems. Moreover, the temporal patterns can be utilized by the system administrators to set up event/incident management rules to make the system more intelligent, and therefore makes autonomous management of large scale distributed systems possible.

Since decades ago, a large portion of the systems have been equipped with mature logging module [RT74, Joh89, ALGJ99, Bro99], But embarrassedly, for a very long time, the recorded events were put aside and thus largely underutilized. In early years, if the events need to be investigated (e.g., system continuously work abnormal), people used to analyze the logs manually by using commands such as `head`, `tail`, `grep`, `cut`, and `sed`, etc. To make the analysis easier, in the 1980s, people developed simple script language `awk` [AKW79] to facilitate the complex log analysis. Soon after, the high level dynamic programming language `Perl` [CWO⁺12] has been proposed and it gradually became the mainstream analysis tool for advanced text/log processing, due to its powerful regular expression support.

All these aforementioned methods can be regarded as the rule-based event analysis, in which the analysts define a set of match rules to find the potential problems. They are effective in the early years because, at that time, the amount of events is small and their relationships are relatively simple. However, due to the fast evolving of computing systems, the size of generated event logs grows exponentially every year and the dependencies between the events become increasingly complicated. In nowadays, even the software on personal computer can have thousands of distinct event types. For example, it is known that a PC with Windows XP can easily generate more than 10000 messages with more than 6000 distinct event types [Eve14]. Similar number of messages can be generated by Windows 7. The situation is even more severe for the state-of-the-art distributed systems as they are designed to be more complicated. As reported by Google's researchers [XHF⁺10], a moderate system in Google would generate over 1 billion event messages each day, with a size about 400 GB uncompressed. These events record the running status of the system involving more than 20000 distinct event types. Since the human analyst can easily be overwhelmed when facing such a large number of events with complex correlations, manual way for event analysis is no longer practical. To catch up the pace of event generating, the automatic way is required.

In recent years, the booming of data mining [WF05, HKP06] and machine learning [Mit97, Bis06] has drawn many attentions from the system management community. People began to leverage advanced data mining and machine learning techniques to facilitate the event analysis as well as the system management. In general, several aspects of system management can be benefited if data mining and machine learning techniques are used:

- **Event log analysis:** Event logs analysis is a major part of system administrators' work. By investigating the daily generated logs, the potential risks of the systems, if any, can be timely figured out. Once the threats have been identified

and eliminated, the systems can be kept in normal conditions and the downtime can be reduced. To facilitate manual event log investigation, a lot of convenient tools/services have been proposed, such as *Windows Event Viewer* [Vie], *FirstApp System Tracker* [App], *Splunk Event Log Viewer* [Spl, Car12], and *Loggly Log Management Tool* [Log], etc. They reduce the administrators' daily work by presenting user-friendly GUIs that facilitate the viewing of the event logs. In current stage, these tools only provide basic statistic tools and still require system administrators to find the potential risks/problems manually. It is believed that by incorporating advanced data mining and machine learning techniques, these tools can be enhanced and they can provide administrators more informative comments about the running status of the monitored systems.

- **System Tuning and Maintenance:** In early years, the systems are tuned and maintained manually by experienced system administrators. They need to periodically investigate the running status of the systems to make sure that all the servers are working properly. The workload is affordable when the number of managed servers is small. However, with the increasing demand for computing in recent years, today's IT infrastructure can easily scale up to hundreds even thousands of servers. In the information era, even a small sized institution would require at least a hundred of servers, which needs a team of system administrators for routine maintenance. To reduce the ever-increasing burden for system administrators, IT service giant IBM proposed the concept of *Autonomic Computing* [Hor01, KC03, GC03] in the early 2000s. Autonomic computing represents the inevitable evolution of IT automation. Its goal is to enable the systems automatically manage themselves according to the administrators' goals. By adopting autonomic computing, systems can be self-adapted in most of the time and the human intervention is needed only in rare situation, resulting significant efficiency improvement for system administrators. In

principal, to fulfill autonomic computing, advanced data mining and machine learning techniques are required.

- **System anomaly detection:** Detecting the abnormal behavior of the system is one of the most critical tasks for system management. To maximize the up-time of systems, administrators need to continuously monitor the running status of the systems to ensure that the systems are in normal condition. As event logs are too large to examine line by line manually, administrators typically create *ad hoc* scripts to search for keywords such as “critical”, “fatal”, or “error”, but this has been shown to be far from enough for problem determination [OS07, JHP⁺09, Xu10]. Rule-based solution is useful when the internal logic of the systems is simple [Pre03], but the knowledge requirement for setting rules increases vastly as the systems are getting more and more complex. To truly conduct the intelligent anomaly detection, advanced data mining and machine learning techniques are necessary to enable the self-learning and self-diagnosis of the systems.

To enhance the aforementioned aspects of system management via data mining and machine learning, the following techniques can be leveraged:

- **Frequent pattern mining:** In data mining community, frequent pattern mining [HCXY07] has been studied for decades. This research direction was first proposed by Agrawal [AIS93] to address the problem of discovering the frequently appeared itemsets in transactional dataset. After that, abundant literatures [HPY00, HPMA⁺00, PHM⁺00, PPC⁺01, YH02, GHP⁺03, HPYM04] have been dedicated to this research area and tremendous progress has been made. These works focused on proposing solutions on various kinds of extensions and applications. In its application in the domain of system management, frequent pattern mining are mainly used to find the frequently co-

occurred events from the logs, including web log mining [CMS97, ZL01, IV06] and access pattern mining [PHMAZ00, ZHF06, PG07]. To reduce the number of discovered patterns, usually maximal patterns [GZ01, BCF⁺05] or closed patterns [Vaa04, ZH05, LOP06] are mined instead of the exhaustive patterns.

- **Sequential pattern mining:** Similar to frequent pattern mining, sequential pattern mining [ZB03, ME10] also focuses on finding frequently appeared temporal patterns. Instead of finding such patterns from a large number of transaction itemsets, sequential pattern mining algorithms are used to discover the patterns from a single but very long event sequence. Compared with frequent pattern mining, sequential pattern is a more natural way to express the occurrence of the events in a system. In recent years, a lot of works have been proposed to conduct system analysis using sequential pattern mining techniques [JD02, LKL07, SG08, HC08]. Also, to reduce the number of mined patterns, the methods of discovering closed sequential patterns [WH04, ZLC10, TC12] are studied. Moreover, variations like partial episodes mining [LB00, LBV01, DP07] and constraint-based episodes mining [PHW02, MR04, PMSR09] are also been proposed to address the problems with special requirements.
- **Time series prediction:** The problem of time series prediction, *a.k.a. time series forecasting*, has been studied by people from different research areas [Aka69, PR81, PP88, Har90, Bro04, BJR13] for decades. Due to its inherent difficulty, this problem is still challenging and new approaches are proposed each year. From the modeling perspective, a lot of models have been proposed to address the prediction problem, including kalman filter [Har90, JU97, LGS⁺08], autoregressive model [Aka69, LR85, Ing03], regression model [Ped97, Lew00, PIL06], neural network [TdAF91, KR94, KB96, FC98, CYD06], support vector machine [SS09], and genetic algorithm [KH00, ZTL⁺04], etc. Also, a couple of learning methods have been leveraged to learn the parameters of the model, including least

square [VGSB⁺01, Abd03, Zha04], gradient descent [Fri01, DJCR01], simulated annealing [Ing93, PH05, PH06], and randomized algorithm [DGHS95, Sch98], etc. In the domain of system management, time series prediction techniques are mainly used to predict the future behavior of the systems. The prediction results can be leveraged to adjust the systems for better reaction of future demands.

- **Outlier detection:** Outlier detection is one of the most attractive research areas in data mining and machine learning [CBK07, CBK09]. Many approaches have been proposed in recent years [AY01, HHWB02, AP02, PKGF03, LK05], these methods leveraged various kinds of information, such as attribute values, data distribution, and context information, to identify the outliers in a data-driven perspective. In general, three types of outliers have been studied: 1) *point outlier*, which is defined as the instance whose attribute values are different from the values of the normal ones; 2) *contextual outlier*, which is defined as the instance whose attribute values are abnormal given a specific context; and 3) *collective outlier*, which is defined a subset of instances altogether are considered as abnormal. There are also a lot of efforts on applying outlier detection to systems management, including the areas of intrusion detection [BCH⁺01, GTD-VMFV09], malware detection [YWLY07, YWL⁺08, YLJW10], and system monitoring based anomaly detection [LX01, MTDB04],

1.2 Contribution

In this dissertation, we would address the research challenges that covers the aforementioned directions. Concretely, we would focus on designing and developing data-driven solutions to help system administrators better manage the systems, including (1) event summarization algorithms to help people better understand the underlying

system event relationships; (2) robust temporal prediction algorithms to help plan the resource capacity by predicting the future demands; and (3) machine learning approaches to help identify the potential system anomaly in real time. In particular, we make the following contributions in this dissertation.

1.2.1 Effective Algorithm and Framework for Event Summarization

The first contribution is generally focusing on the direction of leveraging data mining and database techniques to design an effective algorithm for event summarization. The proposed algorithm is able to summarize an event sequence using *inter-arrival histograms* by capturing the temporal relationship among events. Specifically, the contributions for this area are:

- We propose a systematic and generic framework for natural event summarization using *inter-arrival histograms*. Through natural event summarization, an event sequence is decomposed into many disjoint subsets and well-fitted models (such as periodic patterns and correlation patterns) are used to describe each subset. Inter-arrival histograms demonstrate clear event patterns and capture temporal dynamics of events.
- We formulate the event summarization problem as *finding the optimal combination of event patterns using MDL principle [Grü07]*. The usage of MDL principle automatically balances the complexity and the accuracy of summarization and makes our framework parameter free.
- We present the *encoding scheme* for histograms and cast the summarization problem as seeking a shortest path from the histogram graph.

- We propose a boundary pruning algorithm that greatly accelerates the event summarization process by taking advantage of the multi-resolution property of wavelet transformation.

1.2.2 A Comprehensive Framework for Making the Cloud More Intelligent

The second contribution is a novel and comprehensive framework to model and predict the future demands and capacity of the cloud systems. By making use of the techniques of data mining and machine learning, this framework is able to make the cloud platform more efficient in terms of service fulfillment time and resource usage. Specifically, the contributions for this direction is summarized as follows:

- We formulate both the problem of instant Virtual Machine (VM) provisioning as the time series problem. And we introduce an asymmetric and heterogeneous measurement called *Cloud Prediction Cost* in the context of cloud service to evaluate the quality of our prediction results.
- We design an integrated system that predicts the future demands by combining the prediction power of a set of state-of-the-art algorithms. Our system is able to predict the future demand of each VM type. Especially, based on the temporal dynamics of the VM type importance, our system is able to dynamically take different pre-provisioning strategies.
- We conduct a series of simulation experiments on real dataset to demonstrate the effectiveness of our system.

1.2.3 Online Anomaly Detection Solution on Distributed Systems

The third contribution is a framework to discover a new type of anomaly called *contextual collective anomaly* over a collection of data streams in real time. A primary advantage of this solution is that it can be seamlessly integrated with real time monitoring systems to timely and accurately identify the anomalies. Also, the proposed framework is designed in a way with a low computational intensity, and is able to handle large scale data streams. Concretely, the contributions can be described as follows:

- We provide the definition of contextual collection anomaly and propose an incremental algorithm to discover this type of anomalies in real time. The proposed algorithm combines the contextual as well as the historical information to effectively identify the anomalies.
- We propose a flexible three-stage framework to discover such anomalies from multiple data streams. This framework is designed to be distributed and can be used to handle large scale data by scaling out the computing resources. Moreover, each component in the framework is pluggable and can be replaced if a better solution is proposed in the future.
- We empirically demonstrate the effectiveness and efficiency of our solution through the real world scenario experiments.

1.3 Chapter Summary

In spite of the fact that system events are useful information for system inspection, management, and diagnosis, they has not been made good use by the system administrators today. In this dissertation, we would discuss the usage of event log in three

application domains in system management, and we also present several approaches that use the events to facilitate the system administrators.

To facilitate the reading and understanding, we hereby give an outline of the materials presented in this dissertation. In the next chapter, we would firstly state the problems we would address in this work. In Chapter 3, we will study the problem of leverage data mining techniques, especially the temporal mining techniques, to conduct the summarization task for a given piece of system events. Then in Chapter 4, we will focus on how to leverage time series prediction techniques to make the cloud platform more effective. More specifically, we mainly discuss two closely related problem – cloud capacity planning and VM provisioning prediction. Afterwards, in Chapter 5, we will focus on the problem of how to timely and accurately identify the anomalies among a set of event streams. In this chapter, we would propose a comprehensive framework that leverages advanced data mining and machine learning techniques to identify the contextual collective anomalies among multiple event streams in real time. Finally, we will conclude my research in Chapter 6.

CHAPTER 2

Related Work

In this chapter, we would highlight the research efforts that are related to the three directions that will be addressed in this dissertation. In particular, Section 2.1 presents the existing works on event summarization and the relevant techniques that are used in this problem; Section 2.2 reviews the existing work of system oriented data mining and system behavior modeling; Section 2.3 describes the existing approaches of stream anomaly detection.

2.1 Related Work of Event Summarization

In this section, three related areas that are related to our proposed solution will be discussed: 1) The existing works of event summarization, which is directly related to the problem we intend to solve. 2) The existing application of *MDL* principle in data mining, which is a very useful principle to guide the model selection during our summarization process. and 3) The existing applications of wavelet transformation in data mining, which is an elegant multi-resolution analysis tool and we use it to accelerate the summarization process.

2.1.1 Event Summarization Algorithms

Event summarization is a relative new research area that combines the area of data mining and computer systems. It can be deemed as an extension of frequent itemset mining [AMS⁺96, CSD98] and frequent episodes mining [CS02b, LSU05, MH01, MS01]. These frequent pattern mining techniques can reveal some interesting patterns by identifying the correlations of discrete events and are the building blocks of event summarizations. Event summarization has already attracted a lot of research attentions in recent years [KT08, KT09, PPLW07, WWLW10]. Peng et al. [PPLW07]

proposed an approach to find the patterns in the event logs by first measuring the events inter-arrivals patterns, and then the statistical test is leveraged to identify whether the pairwise temporal correlations do exist. By assembling the correlated event pairs, a correlation graph can be generated to represent the relationship of the events. Different from the work of [PPLW07], Kiernan et al. [KT08,KT09] proposed a summarization method by segmenting the event sequences according to the frequency changing of the events. They leveraged the *MDL* principle to encode the frequency of the events locally and globally. Based on the encoding scheme, they designed a dynamic programming based algorithm to discover the best representation of the event sequences from the perspective of frequency changing. Based on the work of Kiernan et al., Wang et al. [WWLW10] further improved the summary by proposing a *Hidden Markov Model* to describe the transition of states among sequence segments.

It can be seen that several summarization models have been proposed, but these models are either too general or too hard to be understood by data mining outsider. Therefore, none of the results provided by these work are helpful enough for the system administrators.

2.1.2 Minimum Description Length Applications in Data Mining

Minimum Description Length Principle (*MDL*) [Grü07] is a general method for inductive inference. It is effective for generalization, data compression, and de-noising. In data mining community, it has been successfully used for temporal pattern mining [CSD98], clustering [WK90], graph mining [CH94], trajectory outlier detection [LHL08] and social network analysis [XTL⁺10]. Although *MDL* has been used in event summarization for encoding the event segments [KT08,WWLW10], our method is the

first one that uses *MDL* to encode the inter-arrival histogram and to identify the set of disjoint histograms for summarization.

2.1.3 Wavelet Transformation Applications in Data Mining

Wavelet transformation is a useful tool for dividing up data, functions, or operators into different frequency components and then studies each component with a resolution matched to its scale [D⁺92, LLZO02]. This technique has been widely used in many data mining tasks such as clustering [SCZ98], similarity search [PM02], and visualization [MWBF98]. The most relevant application to event summarization is wavelet-based time-series analysis [CF99, PM02] where wavelet tools are used to measure the similarity of time-series in a global perspective. Different from the existing works, in the proposed work, wavelet transformation is used as a subroutine to find the possible boundaries of the patterns hidden in an event sequence, and its multi-resolution property is leveraged to improve the efficiency of our approach.

2.2 Related Work of Cloud Demand Prediction

There are mainly three areas related to this research topic: 1) System oriented data mining; 2) System behavior modeling and prediction; 3) Virtual environment management. In the following, we will discuss these related works in detail.

2.2.1 System Oriented Data Mining

The increasing complexity and scale of modern computing systems make the analytic difficulty far beyond the capability of human being. The emergence of system auxiliary technologies liberates the heavy burden of system administrators by leveraging the state-of-the-art data mining technologies. There are mainly two cat-

egories of system auxiliary technologies: the system analytical techniques and the system autonomous techniques. A couple of efforts have been paid towards these areas. For the first type, [PLM05, LPP⁺10, LLMP05] utilize text mining to find out the category of events and then exploit visualization techniques to show the results. [KT09, WWLW10, PPLW07] utilize temporal mining and encoding theory to discover the event interaction behaviors from systems logs and then summarize them in a brief way. Xu et al [XHF⁺09, XHF⁺09, XHF⁺08] focuses on proposing anomaly detection algorithms to detect the system faults by utilizing the system source code and logs. These above methods are all off-line algorithms and are unable to tell the system to take reactions on-the-fly. For the second type, [PMSR09] uses motif mining to model and optimizes the performance of data center chillers. [GRGM09] proposes a signature-driven approach for load balance in the cloud environment with the help of utilization data. In this thesis, our proposed solution can be categorized as an system autonomous technique. Different from the existing works, we tackle the capacity planning and instant VM provisioning problems from the angle of entire cloud system.

2.2.2 System Behavior Modeling and Prediction

In operating system, caching is one of the common techniques used to improve the system performance through forecasting. *Partitioned Context Modeling* (PCM) [KL99] and *Extended Partitioned Context Modeling* (EPCM) [KL01] are two statistical based caching algorithms, which model the file accesses patterns to reliably predict upcoming requests. These two methods have shown much better performance over the traditional caching algorithms like *Least Recent Used* (LRU), *Least Frequent Used* (LFU) and their extensions. While in the area of modern large-scale system behavior prediction, there is only little related works. Different from the traditional shared

memory scenario, the virtual machines in the cloud environment can not be shared and reused due to the security issue. Therefore, there should be multiple copies of virtual machines prepared for multiple requests. The work of [GGW10] also studies the cloud resource prediction problem. But it only focuses on predicting the VM resource (CPU, memory, etc) for individual VMs. For my work, rather than predicting the resource usage within VMs, I aim to predict the capacity and VMs demands for the whole cloud.

2.2.3 Virtual Environment Management

Virtual environment management technologies are mainly based on control theory, where the actions are adjusted according to the previous status of the systems. Auto-scaling proposed by Amazon [MLH10] is the customer side auto-scaling management component, which allow customers to set up their own rules to manage the capacity of their virtual resources. This method focuses on the customer side post-processing of capacity tuning rather than the provider side. In [MSB10], the resources allocation problem is modeled as a stochastic optimization problem with the objective of minimizing the number of servers as well as the SLA penalty. [MLS10] proposes a technique to enable the auto-scaling of visualized data center management servers. All these works are focusing on the resource allocation and scheduling with a fixed amount of resources, while in this thesis, the proposed method aims at estimating the total amount of resources for the whole cloud environment.

2.3 Related Works of Stream Anomaly Detection

Although anomaly detection has been studied by researchers for years [HA04,CBK09]. To the best of our knowledge, our proposed method is the first one that focused on mining contextual collective anomalies among multiple streams in real time. In this

section, we briefly review two closely related areas: mining outliers from data streams and mining outliers from trajectories.

With the emerging requirements of mining data streams, several techniques have been proposed to handle the data incrementally [ZS02, JPLC12a, JPLC12b, TTD⁺08, JPLC11]. Pokrajac et al. [PLL07] modified the static Local Outlier Factor (LOF) [BKNS00] method as an incremental algorithm, and then applied it to find data instance anomalies from the data stream. Takeuchi and Yamanishi [TY06] trained a probabilistic model with an online discounting learning algorithm, and then use the training model to identify the data instance anomalies. Angiulli and Fassetti [AF07] proposed a distance-based outlier detection algorithm to find the data instance anomalies over the data stream. However, all the aforementioned works focused on the anomaly detection of a single stream, while our work is designed to discover the contextual collective anomalies over multiple data streams.

A lot of works have been conducted on trajectory outlier detection. One of the representative work on trajectory outlier detection is conducted by Lee et al. [LHL08]. They proposed a partition-and-detection framework to identify the anomaly sub-trajectory via the distance-based measurement. Liang et al. [TTJ⁺08] improved the efficiency of Lee’s work by only computing the distances among the sub-trajectories in the same grid. As the aforementioned two algorithms require to access the entire dataset, they cannot be adapted to trajectory streams. To address the limitation, Bu et al. [BCFL09] proposed a novel framework to detect anomalies over continuous trajectory streams. They built local clusters for trajectories and leveraged efficient pruning strategies as well as indexing to reduce the computational cost. However, their approach identified anomalies based on the local-continuity property of the trajectory, while our method does not make such an assumption. Our approach is close to the work of Ge et al. [GXZ⁺10], where they proposed an incremental approach to maintain the top-K evolving trajectories for traffic monitoring. However, their

approach mainly focused on the geo-spatial data instances and ignored the temporal correlations, while our approach explicitly considers the temporal information of the data instances.

Temporal Mining for Event Summarization

In this chapter, we would mainly focus on the problem of providing effective event summarization methodologies to facilitate the system inspection and analysis for the system administrators. Part of the content in this section has been published during my Ph.D study, including the problem formulation and the proposed solutions.

The outline of this chapter is as follows: The motivation and the research objective of this problem will be introduced in Section 3.1. In Section 3.2, a novel event summarization algorithm will be presented and the corresponding experimental evaluation will be presented. Then in Section 3.3, a multiresolution event summarization framework will be introduced and evaluated. Finally, a short chapter summary about this problem and the proposed solutions will be given in Section 3.4.

3.1 Motivation

Given the event logs, event mining is a useful way to understand the behaviors of the computer systems. In previous years, most existing event mining research efforts focused on episodes mining or frequent pattern discovering.

These simply output a number of patterns that are independent to each other, so they are unable to provide a brief and comprehensive event summary revealing the big picture that the dataset embodies.

Instead of discovering frequent patterns, recent works on event mining have been focused on event summarization, namely how to concisely summarize temporal events [KT08, KT09, WWLW10]. Current state-of-art event summarization techniques are based on sequence segmentation. These methods first split the event sequences into disjoint segments and then the patterns are generated and summarized to describe events in each segment. Kiernan and Terzi [KT08, KT09] model the set segmentation problem as an optimization problem that balances between the shortness of the local models

for each segment (i.e., the summary) and the accuracy of data description. Their method reveals the local patterns of the sequence but does not provide the inter-segment relationships. Based on their work, Wang et al. [WWLW10] provide a more sophisticated method that not only describes the patterns in each segment but also learns a Hidden Markov Model (HMM) to characterize the global relationships among the segments.

3.1.1 The Need for A More Intuitive Event Summarization

Although there are a number of existing solutions for event summarization, but some important aspects of the events still cannot be demonstrated in the summarization results. The following motivating example reflects such shortages.

Example 1. *Figure 3.1 shows an event sequence containing 4 event types, where event type A denotes “an event created by an anti-virus process”, event type B denotes “the firewall asks for the access privilege”, event type C denotes “a port was listed as an exception”, event type D denotes “the firewall operation mode has been changed”. The approach presented in [WWLW10] segments the sequence into 4 segments based on the frequency changes of the events. For each segment, the approach further clusters the event types according to the frequency of each event type. Finally, an HMM is used to describe the transactions between intervals.*

The result generated by [WWLW10] partially solves the problem of frequent pattern mining solutions by providing a comprehensive summary for the input sequence. However in principle, this method is a frequency-based method and such category of methods have several limitations:

- The frequency-based summarization methods only focus on the frequency changes of event types across adjacent segments and ignore the temporal information

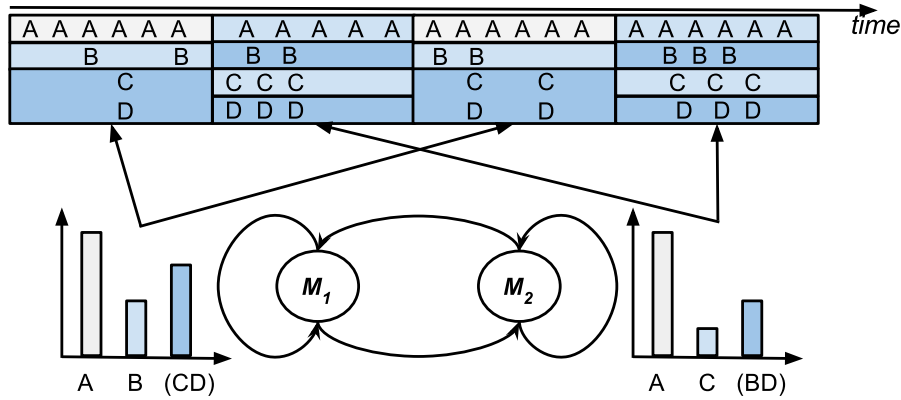


Figure 3.1: A motivating example for event summarization

among event types within a segment. As a result, the generated summary fails to capture the temporal dynamics of event patterns. It is known that due to the dynamic workload of the systems, the frequency of event occurrences cannot reveal too much information about their running status. It is trivial to know that the frequency of a certain event types would be high when the workload of the system is high, and vice versa.

- The frequency-based summarization methods generate the same number of event patterns with the same boundaries for all event types. The unified pattern boundary requirement is impractical since different event types can have different underlying generating mechanisms. Consider in a distributed system, event may come from thousands of hosts that are not related to each other, forcing global segmentation means a simple event episode can be split to many segments because other events do not occur uniformly during the period; this yields a large number of small segments that may unnecessarily inflate the event summary.
- The generated summary is represented in form of abstruse mathematical model. Such kind of summary is acceptable for people with strong math background

but is not simple enough to be used by all system administrators. Without clear understanding of the summary, system administrators can hardly extract useful information from the summary and take proper actions accordingly.

3.1.2 The Need for a General Event Summarization Framework

As previously mentioned, several research efforts have been working on providing various summarization methods. Each of these works define its own way of summarizing representation. On the other hand, there are some efforts [PTG⁺03] working on providing various techniques for presenting event summarization results. From all these explorations, it is not difficult to conclude that event summarization is not a problem that can be handled by a single model or algorithm. For different purposes or for different users, there are a large number of ways for event summarization, and also many parameters to tune. To obtain an event summary from different perspectives, an system administrator has to re-preprocess the data and change the program time after time. This is a drain of administrators' productivity.

This predicament is very similar to that, in the 1970's, every data-intensive task has to write a redundant program for data manipulation. The appearance of ER model and SQL finally addressed the data representation and query problem and significantly reduced the complexity of data-driven systems. Following the history path of DBMS and query languages, it is not difficult to show that event summarization (as well as event analysis) also need a general and unified data model and a corresponding query language.

The data model and query language should be flexible enough so that the real life scenarios can be efficiently and adequately handled. The followings are some typical scenarios that an event analyst should encounter.

Scenario 1. *An analyst obtains a system log of the whole year, but he only wants to view the summary of the events that are recorded between the latest 30 days. Moreover, he wants to see the summary without the trivial event firewall scan. Also, he wants to see the summarization with the hourly granularity.*

Scenario 2. *After viewing summarization results, the analyst suspects that one particular time period of events behaves abnormally, so he wants to conduct anomaly detection just for that period to find out more details.*

Scenario 3. *The system has generated a new set of security log for the current week. The analyst wants to merge the new log into the repository and also to summarize the merged log with the daily granularity.*

To handle the work in the first scenario using existing event summarization methods, we need to perform the five tasks: (1) Extract the events occurred during the specified time range; (2) Remove the irrelevant event types; (3) Aggregate the events by hour; and (4) Feed the pre-processed events to existing event summarization methods to obtain the summary.

Similarly, about the same amount of works are needed for the second and third scenarios. In the above scenarios, each of the steps requires a separate program to conduct. In usually case, programs need to be created from the scratch to cater the highly customized needs. Also note that, if parameter tuning is needed, a typical summarization task requires hundreds of such iterations. Therefore, a practical event summarization is highly inconvenient and time-consuming.

Similar to Online Analytical Processing (OLAP) as an exploration process for transactional data, event summarization is also a trial-and-error process for temporal event data. As event summarization requires repetitive exploration of the events from different views, It is not difficult to see *the necessity of having an integrated framework to enable users to easily, interactively, and selectively extract, summarize,*

Many action rules can be derived almost directly from the generated summary. Figure 3.2 shows the output summary generated by our natural event summarization for Example 1. In the example event sequence, event with type C , D always appear after B for a short delay in both the second and fourth segments. Similarly, events with type C , D always appear after B for a long delay in both the first and third segments. Therefore, two correlation patterns: $B \rightarrow C$, and $B \rightarrow D$ exist. The change of periods implies that ill-configuration of the firewall may exist. For events with type A , they appear regularly throughout the whole sequence, so all the instances with event type A should belong to only one segment. As event type A is the anti-virus monitoring process event, its stable period can indicate that the anti-virus process works normally.

The framework of natural event summarization is shown in Figure 3.3. Our framework is based on inter-arrival histograms which capture the distribution of time intervals between events. These histograms provide a concise way to describe periodic patterns (of the same event type) and correlation patterns (of different event types). In this preliminary work, we only focus on summarizing on the aspect of temporal patterns of the events, since such information tells most of the story of the running status of the system.

The event summarization problem is formulated as finding a set of disjoint inter-arrival histograms, each representing a certain periodic pattern or correlation pattern, to approximate the original input sequence using *MDL*. *MDL* is an elegant theory to naturally balance the accurateness and the conciseness of the summarization information. Although *MDL* has been used in previous event summarization methods for encoding the event segments, it is used here for encoding the inter-arrival histograms and for identifying the set of disjoint histograms for summarization. The problem of finding a set of disjoint histograms can be solved optimally in polynomial time by seeking a shortest path from the histogram graph that represents the

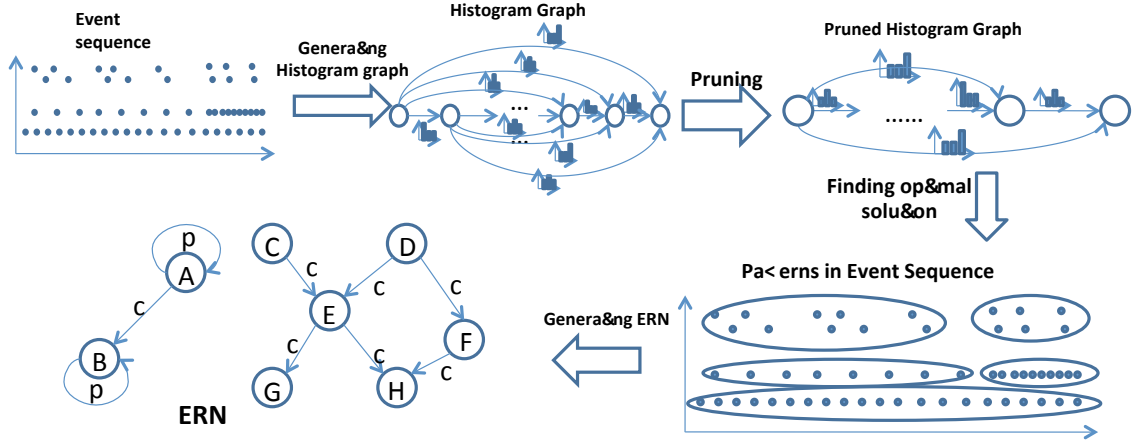


Figure 3.3: The framework of natural event summarization

temporal relations among histograms. To improve the efficiency of event summarization, I also explore an efficient alternative by using the multi-resolution property of wavelet transformation to reduce the size of the histogram graph. The final summary of our natural event summarization can be described as an easy-to-understand event relationship network (*ERN*) where many actionable rules are readily available.

3.2.2 Definitions and Problem Formulation

In this section, we will first give the definitions that will be used to solve this problem will be given first. Then this problem will be formulated.

Interval Histograms

An event sequence D comprises of a series of event instances in the form of (e, t) ordered by their timestamps: $D = (< t_1, e_1 >, \dots, < t_n, e_n >)$, where t_i is a timestamp and e_i is the type of the event. Each instance belongs to one of the m types $\mathcal{E} = \{e_1, \dots, e_m\}$. We first describe the inter-arrival histograms used in our framework to represent the inter-arrival distribution of time interval between events.

Given two event types x and y , let S be the subsequence of D that only contains events of types x and y . Suppose S is split into k disjoint segments $S = (S_1, S_2, \dots, S_i, \dots, S_k)$. I use an interval histogram (or inter-arrival histogram) $h_{xy}(S_i)$ to capture the distribution of time interval between events of type x and type y in S_i . Specifically, the bin $h_{xy}(S_i)[b]$ is the total number of intervals whose length is b . Let $next(t, y)$ denote the timestamp of the *first* type y event that occurs after t in S_i .

Definition 3.2.1. Inter-arrival histogram:

$$h_{xy}(S_i)[b] = |\{i | e_i = x, next(t_i, y) - t_i = b\}|.$$

where t_i denotes the timestamp of e_i .

If $x \neq y$, then the inter-arrival histograms capture the time intervals for the events of different types; for the case of $x = y$, they capture the time intervals of events of the same type. Given an interval histogram, we can use a standard histogram to approximate it. The standard histogram is formally defined with definition 3.2.2.

Definition 3.2.2. Standard Histogram: *Standard histogram is a special kind of interval histogram with one or two non-empty bin and all these non-empty bins have the same value $\frac{\#intervals}{n_{non}}$, where $\#intervals$ indicates the number of intervals and n_{non} indicates the number of non-empty bins. We use $\bar{h}_{xy}(S_i)$ to denote the corresponding standard histogram of $h_{xy}(S_i)$.*

Note that the two types of event relationships: *periodic patterns* and *correlation patters* can be easily described using standard histograms. The *periodic patterns* and *correlation patters* is formally defined in definition 3.2.3.

Definition 3.2.3. Periodic pattern and Correlation pattern: *A pattern is a 5-tuple (t_s, t_e, x, y, P) , where (1) t_s and t_e denotes the start position and end position of the event sequence described by the pattern respectively. (2) x and y denote the*

types of events involved in the pattern, (3) P contains the periodic parameters. The pattern can contain 1 or 2 period parameters, which indicate the inter-arrival value between event x and y . Moreover, if $x = y$, this pattern is a **periodic pattern**, otherwise, it is a **correlation pattern**.

Example 2 provides a detailed illustration to show how to utilize standard histogram and periodic/correlation patterns to summarize a given event sequence.

Example 2. Given an event sequence D , the timestamps for e_a , e_b and the related subsequences are listed in Table 3.1 (S for event type a, b is the same as D). There is only one segment in S and there exists a periodic pattern for e_a and a correlation pattern between e_a and e_b . The segment are described by inter-arrival histograms $h_{aa}(S)$ and $h_{ab}(S)$ in Figure 3.4 and are approximated by two standard histograms in Figure 3.5.

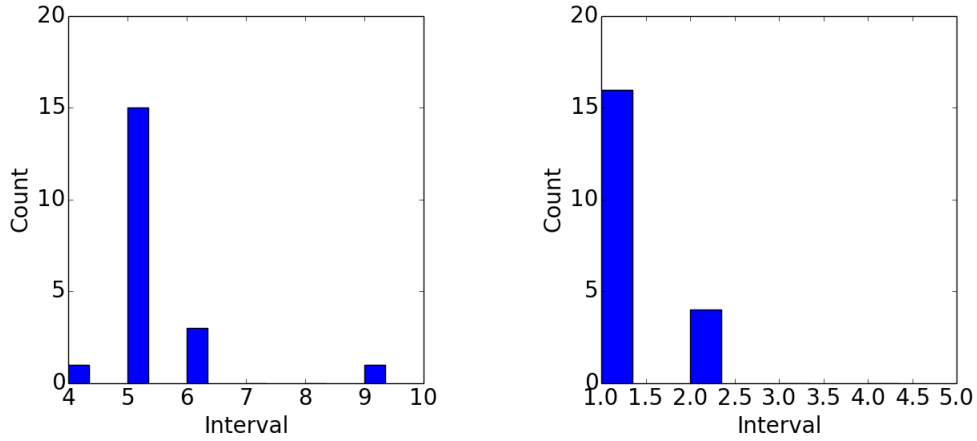


Figure 3.4: (a) Inter-arrival histogram $h_{aa}(S)$; (b) Inter-arrival histogram $h_{ab}(S)$.

Table 3.1: Occurrence of two events

Event type	Occurrence timestamp
a	5,11,16,21,26,30,35,41,46,51,61,66,71,76,81,86,92,97,102,107
b	6,12,17,22,28,31,36,43,47,52,63,67,72,77,82,87,93,99,103,108
S for $h_{aa}(S)$	The same as a
S for $h_{ab}(S)$	5,6,11,12,16,17, ..., 102,103,107,108

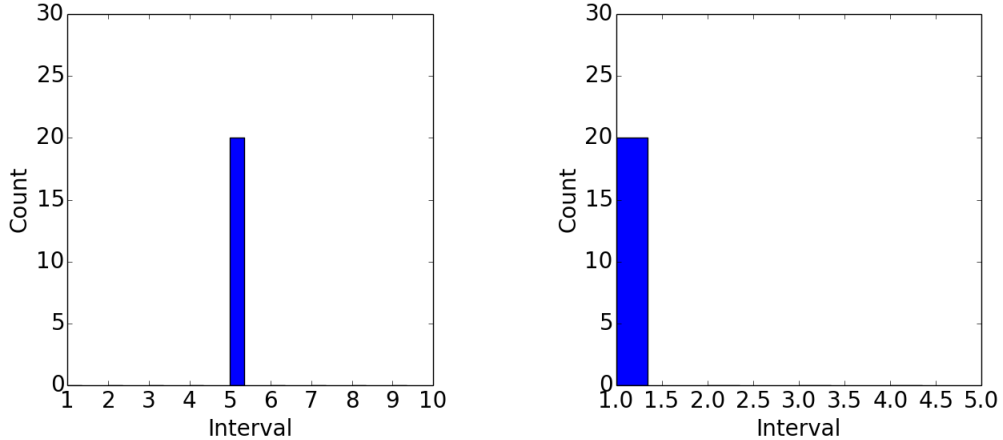


Figure 3.5: (a) Standard histogram that best approximate $h_{aa}(S)$; (b) Standard histogram that best approximate $h_{ab}(S)$.

It is obvious that the histograms demonstrate clear patterns. However, the total number of different histograms can be large for datasets with many events. The 2 histograms shown in Figure 3.4 and their corresponding standard histograms in Figure 3.5 are just one of many possible histogram combinations that depict the event relationships/patterns hidden in Example 2. For example, two histograms can be used to respectively depict the first and the second half of e_a . Hence one challenge is how to find the most suitable combination of histograms to describe the event sequence. In our framework, *MDL* principle is leveraged to find out the best set of disjoint segments and the corresponding standard histograms to summarize the event sequence.

3.2.3 Summarization using MDL

In this section, we propose an information theoretic method to describe the histogram, i.e., encoding the histogram in *bits*. Marsland et al. [MTT08] proposed an encoding scheme for histograms with fixed number of bins and fixed number of bin elements (the total value of all bins). However, for the histograms used in event summarization, neither the number of bins and the number of bin elements is fixed beforehand. To the

best of our knowledge, there is no encoding scheme that can be directly used in our scenario. Based on such situation, we first describe the scheme for encoding standard histograms (e.g., periodic patterns and correlation patterns). Then given an inter-arrival histogram, we show how it can be approximated using standard histograms. Finally, we formulate the event summarization problem. It should also be pointed out that, although *MDL* has been used in event summarization for encoding the event segments in a few works [KT08, WWLW10], our work is the first one that uses *MDL* to encode the inter-arrival histograms and to identify the set of disjoint histograms for summarization.

Encoding Standard Histograms

Given an event subsequence S of event types x and y with disjoint segments $S = (S_1, S_2, \dots, S_i, \dots, S_k)$, to encode a standard histogram $\bar{h}_{xy}(S_i)$, the following four components need to be encoded. These components are necessary and enough to describe the histogram.

Event type depicted by the histogram. Each histogram should be associated with one (for periodical patterns) or two (for correlation patterns) event types depending on the type of the relationship it depicts. Given the set of event types \mathcal{E} , it is sufficient to use $L(m) = \log |\mathcal{E}|$ bits to represent each event type.

Boundaries of S_i . The relationship depicted by an interval histogram has two boundaries indicating the start and end positions of the corresponding segment. Each boundary requires $L(b) = \log |S|$ bits to encode its information.

Information of non-empty bins in the histogram. This piece of information can be encoded using

$$L(bin) = \log \delta + \log i_{max} + \log |S| + \sum_{i=1}^{n_{non}} \log i_{max} + \sum_{i=1}^{n_{non}} \log |S_i|.$$

The coding length consists of five terms. The first term uses $\log \delta$ bits to encode the largest interval i_{max} in S_i , where δ denotes the allowed largest interval. In our framework it is set as the number of seconds in one day. The second term uses $\log i_{max}$ bits to encode the number of non-empty bins n_{non} . The third term uses $\log |S|$ bits to encode the length of S_i . The fourth and fifth terms encode all the indices (1 to i_{max}) and the number of elements contained in the n_{non} bins respectively.

Putting them all together, the bits needed for encoding a standard histogram $\bar{h}_{xy}(S_i)$ is

$$L(\bar{h}_{xy}(S_i)) = L(m) + L(b) + L(bin). \quad (3.1)$$

Encoding Interval Histograms

Given an inter-arrival histogram $h_{xy}(S_i)$, a quantitative criteria is needed to measure how well it can be represented by event patterns, or equivalently, how well it can be approximated by standard histograms.

Histogram distance. Histogram distance describes how much information is needed to depict the *necessary bin element movements* defined in [CS02a] to transform $\bar{h}(S_i)$ into $h(S_i)$ (To make notations uncluttered, we drop the subscripts xy and they should be clear from the context.). The code length of the distance can be calculated as:

$$L(h[S_i]|\bar{h}[S_i]) = \sum_{i \in Non} |be_i - bs_i| \log i_{max}, \quad (3.2)$$

where Non is the union of the index sets of non-empty bins in both histograms, be_i and bs_i denote the number of elements in bin i in histogram $h[S_i]$ and $\bar{h}[S_i]$ respectively. For each element at bin i , a new bin index can be assigned to indicate where it should be moved. Equation 3.2 measures the bits of information needed by summing up the elements in unmatched bins.

In summary, the amount of information required to describe an inter-arrival histogram $h(S_i)$ using an event pattern $\bar{h}(S_i)$ equals to the summation of the code length for $\bar{h}(S_i)$ and the distance between $h(S_i)$ and $\bar{h}(S_i)$. Since there may be multiple standard histograms, we define the code length for an interval histogram $h(S_i)$ as follows:

$$L(h(S_i)) = \operatorname{argmin}_{\bar{h}(S_i) \in \bar{H}(S_i)} L(\bar{h}(S_i) + L(h(S_i)|\bar{h}(S_i))), \quad (3.3)$$

where $\bar{H}(S_i)$ is the set of all possible standard histograms on S_i .

Problem Formulation

Given an event sequence D , for each subsequence S containing event types x and y , the minimum coding length $L(S)$ for S is defined as

$$L(S) = \operatorname{argmin}_{\{S_1, S_2, \dots, S_k\}} \sum_i L(h(S_i)). \quad (3.4)$$

Since the boundaries for different subsequences are independent, then the minimum description length for the input event sequence D is

$$L(D) = \sum_{S \in D} L(S). \quad (3.5)$$

Hence the event summarization problem is to *find the best set of segments $\{S_1, S_2, \dots, S_k\}$ as well as the best approximated standard histograms to achieve the minimum description length.*

3.2.4 The NES Algorithm

In this section, a heuristic algorithm that can find the best set of segments of S in polynomial time is first introduced. Then the method of generating *ERN* using the event patterns is presented in detail.

Finding the best segmentation

The problem of finding the best set of segments can be easily reduced to the problem of finding a shortest path from the generated histogram graph G . The histogram graph G is generated as follows:

1. Given S , let n_{xy} be the number of inter-arrivals between event x and y (x, y can be the same type) in S , generate n_{xy} vertices and label them with the positions of each x (1 to n_{xy}).
2. Add $edge(a, b)$ from vertex $v[a]$ to vertex $v[b]$ for each vertex pair $v[a], v[b]$, where $1 \leq a < b \leq n_{xy}$. Assign the weight of each $edge(a, b)$ as $L(h(S_i))$, where S_i starts at position a and ends at b . Note that to compute $L(h(S_i))$, we need to find the best standard histogram $\bar{h}(S_i)$ for $h(S_i)$ in the sense that $L(h(S_i))$ is minimized (as shown in Eq.(3.3). This can be done using a greedy strategy. Given $h(S_i)$, we can sort the bins in decreasing order based on their values. Then iteratively perform the following two steps: (1) generating $\bar{h}(S_i)$ using the top i bins, and (2) computing $L(h(S_i))$. (3) increase i by 1. The iteration continues till $L(h(S_i))$ begins to increase. The $\bar{h}(S_i)$ corresponding to the minimum description length $L(h(S_i))$ is often referred as the best standard histogram for $h(S_i)$.

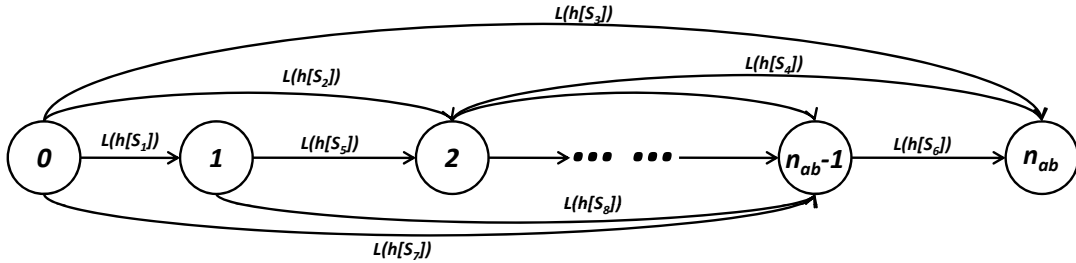


Figure 3.6: An example histogram graph

After generating the histogram graph, we can use the classical shortest path algorithm (e.g., *Dijkstra* algorithm) to output the vertices on the shortest path from $v[1]$

to $v[n_{xy}]$. Figure 3.6 shows an example histogram graph and Algorithm 1 illustrates the *summarization* process of our natural event summarization framework (*NES* for short). In line 4, the algorithm generates a set of event subsequences from D , there are m subsequences for the same event type (i.e., $x = y$) and $m^2 - m$ subsequences for different event types (i.e., $x \neq y$). Line 6 generates the directed acyclic histogram graph for each subsequence S and uses *Dijkstra* algorithm to find the shortest path $P = (v[i_1], v[i_2], \dots, v[i_p])$. The best segmentation solution contains the segments $\langle v[i_1], v[i_2] \rangle, \langle v[i_2], v[i_3] \rangle, \dots, \langle v[i_{p-1}], v[i_p] \rangle$. Line 8 and 9 represent each segment using the best fitted event patterns (i.e., the best standard histograms), and put them into the set \mathcal{R} .

Algorithm 1 The NES Algorithm

1. **input:** event sequence D .
 2. **output:** all relationships \mathcal{R} .
 3. Identify m from S , relationship set $\mathcal{R} \leftarrow \emptyset$;
 4. Separate D into a set of S ;
 5. **for all** S **do**
 6. Generate directed graph G ;
 7. Use **Dijkstra**(G) to find shortest path P ;
 8. Generate relationships R from P ;
 9. $\mathcal{R} \leftarrow \mathcal{R} \cup R$;
 10. **end for**
 11. **return** \mathcal{R} ;
-

For each histogram graph G , the time complexity of *Dijkstra* algorithm is $O(|E| + |V| \log |V|) = O(|S|^2)$. Therefore, the total running time of Algorithm 1 is $O((m + m(m - 1))|S|^2) = O(|D|^2)$.

Generating summarization graph

Although the set of event patterns can be described in text, we show that they can be used to construct an intuitive event relationship network (*ERN*) [PTG⁺03] which produces a concise yet expressive representation of the event summary. *ERN* is a graph model that represents the relationship between event types. The procedure for

building *ERN* is straightforward: it first generates the vertices for each event type involved in any event patterns, and then adds edges for event patterns and stores necessary information (e.g., segmentation, periods and time intervals) onto the edges.

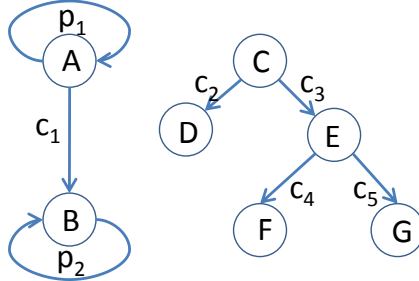


Figure 3.7: An example ERN graph.

Figure 3.7 shows an example *ERN* graph. It contains two periodic patterns: $A \xrightarrow{p_1} A$, $B \xrightarrow{p_2} B$ and five correlation patterns: $A \xrightarrow{c_1} B$, $C \xrightarrow{c_2} D$, $C \xrightarrow{c_3} E$, $E \xrightarrow{c_4} F$, $E \xrightarrow{c_5} G$. For simplicity, the ranges of segments are ignored.

3.2.5 Efficiency Improvement

The NES algorithm described in Section 3.2.4 finds the best segmentation by checking all positions in S , which is computational intensive. In fact, given an event subsequence S , boundaries should only locate at positions where inter-arrival times change rapidly, because segmentation at smooth places would waste encoding bits. Based on this observation, we propose an effective pruning algorithm which is able to prune a large part of boundaries that are unlikely to be the boundaries of segments. This algorithm greatly accelerates the summarization process by reducing the size of the histogram graph. By utilizing the multi-resolution analysis (MRA) of wavelet transformation, the pruning can be done in linear time in the worst case.

Preprocessing

For ease of pruning, some preprocessing steps are needed. Given an event subsequence S , we pre-process it as follows:

1. *Obtaining inter-arrival sequences*: Given $S = (\langle e, t_1 \rangle, \dots, \langle e, t_{|S|} \rangle)$, transform it into inter-arrival sequence $V = (t_2 - t_1, t_3 - t_2, \dots, t_{|S|} - t_{|S|-1})$;
2. *Padding*: Append 0's to the tail of sequence until the length of interval sequence equals to a power of 2;
3. *Transforming*: Use *Haar* [Haa10] wavelet as the wavelet function and apply fast wavelet transformation (FWT) on V to obtain the transformation result W .

The resulting W contains the wavelet coefficients of the inter-arrival sequence, which is the input of our pruning algorithm.

Pruning unlikely Boundaries

Due to the multi-resolution analysis (MRA) property of wavelet transformation, the deviation of inter-arrivals in V can be viewed from W at different resolutions. As wavelet transformation contains the information of both frequency domain and time domain, for each $W[i]$, we can quickly locate the corresponding segment in V . For example, if $|V| = 1024$, the elements $W[1]$ and $W[2]$ contain the averaging information and the *diff* information of the highest resolution (in this example, resolution 1024) of original series respectively. The elements $W[3]$ and $W[4]$ contain the information of V at resolution 512. Specifically, $W[3]$ corresponds to the first half of V and $W[4]$ corresponds to the second half of V .

Taking advantage of such a property, the pruning algorithm is able to identify the inter-arrival deviation in a top-down way. The basic idea of algorithm is as follows: it checks the element $W[i]$, starting from $i = 2$, which contains the deviation of inter-arrival for the whole sequence V . There are three possible cases:

Case I: If $W[i]$ is small, it indicates that the corresponding subsequence in V for $W[i]$ is smooth enough and segmentation is not needed. In this case, the algorithm records the boundaries of the segment and then stops.

Case II: If $W[i]$ is large, it indicates that the deviation of inter-arrivals in the corresponding subsequence in V is too large. In this case, the subsequence needs to be split into two halves and the algorithm needs to perform recursively.

Case III: If $W[i]$ records the deviation of inter-arrivals at the lowest resolution, the algorithm just records the start and end boundaries and returns.

Algorithm 2 Algorithm of *BoundaryPruning*

1. **input:** $W, level, i$.
 2. **output:** Boundary set B after pruning.
 3. Set $B \leftarrow \emptyset$;
 4. $threshold \leftarrow \frac{W[1]}{2^{\log |W| - level}}$;
 5. $spectrum \leftarrow W[i]$;
 6. **if** $spectrum < threshold$ or reaches to the lowest resolution **then**
 7. Add corresponding boundaries to B ;
 8. **else**
 9. $i_1 = 2i - 1, i_2 = 2i$;
 10. $B_1 \leftarrow \text{BoundaryPruning}(W, level - 1, i_1)$;
 11. $B_2 \leftarrow \text{BoundaryPruning}(W, level - 1, i_2)$;
 12. $B \leftarrow B \cup B_1 \cup B_2$
 13. **end if**
 14. **return** B ;
-

Algorithm 2 provides the pseudo-code for *BoundaryPruning*. The parameter $level$ (initialized as $\log |W|$) denotes the current level of resolution that the algorithm checks, and i denotes the current position in transformed result W (i.e. $W[i]$) to be checked. The algorithm calculates the corresponding threshold as $\frac{W[1]}{2^{\log |W| - level}}$, which reflects the average deviation of inter-arrival at resolution 2^{level} and changes dynamically according to the $level$.

Example 3 illustrate how the pruning algorithm works with a toy dataset.

Example 3. An input inter-arrival sequence V is shown in Figure 3.8. The algorithm starts from the highest resolution. It finds that $W[2] = 3$ is too large, so the whole inter-arrival sequence cannot be segmented with only one segment. At a lower resolution, the algorithm finds that $W[4]$ is small enough, so the corresponding subsequence $\langle 2, 1, 2, 1, 2, 1, 3, 1 \rangle$ can be considered as one segment. For the element $W[3] = -2$ representing the first half of V , it is still too large and the corresponding subsequence $\langle 1, 1, 1, 1, 1, 3, 1, 1 \rangle$ cannot be considered as only one segment. Hence the algorithm drills down to a lower resolution to do the same check task. Finally, the algorithm divides V into 5 segments, and 6 boundaries are recorded and the remaining 11 boundaries are pruned. Figure 3.9 shows the effect of BoundaryPruning in reducing the size of the histogram graph.

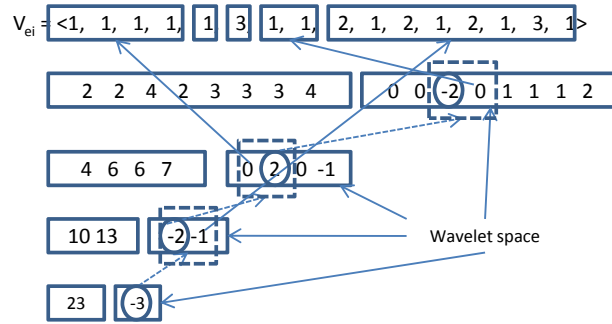


Figure 3.8: Segments information in wavelet spectrum sequence.

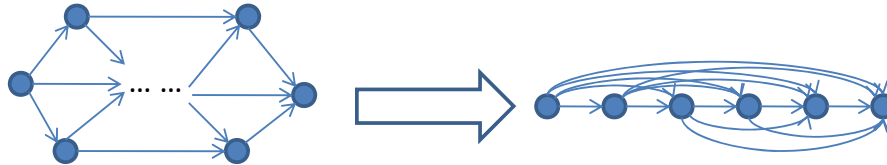


Figure 3.9: Before pruning, the histogram graph contains 17 vertices and 136 edges, after pruning, the histogram graph contains only 6 vertices and 14 edges.

The pruning algorithm only scans W once. In general cases, the algorithm does not need to check all the elements in W . If the inter-arrival sequence is smooth

enough, the algorithm would stop checking at a high resolution. Only in the worst case, the algorithm has to check every elements in W . Since $|W| = |V| = |S| - 1$, the algorithm runs in $o(|S|)$ for the average case and $O(|S|)$ in the worst case.

3.2.6 Experimental Evaluation

In order to investigate the effectiveness and efficacy of the proposed approach, we design two sets of experiments to evaluate our algorithm on both synthetic and real datasets. There are two reasons that we conduct the evaluation with synthetic data:

1. There is no ground-truth summarization result for real world data, thus there is no way to evaluate the summarization result if we directly run the algorithm on real world data.
2. The real world event sequence is generated automatically by the operating system, there is no way to precisely control the size of the dataset. We cannot just pick a subset of the real world sequence with a fixed size, since it may destroy a lot of hidden patterns.

With synthetic data, we can freely set the parameters of dataset such as the number of patterns of each type, the location of each patterns, the level of noise, and the size of dataset. Since we know the ground-truth of the synthetic data, we are able to investigate the capabilities as well as limitations of our algorithm by comparing its output with the ground-truth.

Our task for synthetic data experiments is to answer the following questions: (1) Can our event summarization framework indeed discover the hidden relationships from the event sequence and effectively summarize them? (2) Is our framework efficient in handling event summarization task? (3) Is the approximation algorithm precise enough that covers most of the relationships comparing with the optimal counterpart?

Table 3.2: Parameters and their meanings

Parameter	Description
n	Number of events in sequence.
m	Number of event types.
n_p	Number of periodic patterns.
n_c	Number of correlation patterns.
r_{noise}	Degree of noise.
l	Number of events in one pattern.

For the real world data experiments, there are two questions: (1) Can the proposed algorithm find out any useful patterns from the real world data? (2) Is the summarization result more meaningful and friendly than the counterparts?

Experimental Setup

All the datasets are pre-processed by transforming all event instances into a predefined format $inst = \langle event_type, date, time, source, category, event_ID \rangle$. Since different event types generated by different programs may share the same event_ID in some systems like Windows series OS, it is not enough to distinguish the events just by their ID. We map each distinct tuple $\langle event_type, source, category, event_ID \rangle$ as unique event type e .

Experiments on Synthetic Data

We generate several groups of synthetic datasets to evaluate our approach, each group consists a set of datasets generated by changing a particular parameter and fixing the remains. The meaning of each parameter is listed in Table 3.2. For each dataset, a number of relationships/patterns are intentionally planted and the noise is added according to the by noise level ³ to simulate the real scenario.

³The noise levels are set as the probability of inter-arrival deviation, e.g, if noise level is 0.2, then with the probability of 20% the inter-arrival time will be shifted by a random value.

We use *NES-Prune* to represent summarization with *BoundaryPruning*, *NES* to represent summarization without *BoundaryPruning*.

Accuracy of the approaches. The first goal of experiments on synthetic data is to check whether our approach can really discover the correct hidden patterns from the data. 5 datasets are generated by fixing $n = 15000, m = 150, n_p = 50, n_c = 50, l = 100$ and the noise levels are set from 0.1 to 0.5 with an increment of 0.1. For each dataset, 50 periodic and 50 correlation patterns are generated respectively with distinct period parameters and event types. Then order these patterns are randomly ordered to build the synthetic dataset. Table 3.3 shows the results of how many planted patterns are found via *NES* and *NES-Prune* respectively. In this table, NES_p and NES_c denote the proportion of planted periodic and correlation patterns found by *NES*, $NESP_p$ and $NESP_c$ denote those found by *NES-Prune*.

Table 3.3: Accuracy for synthetic datasets, $n = 15000, m = 150, n_p = 50, n_c = 50, l = 100$

No.	NES_p	NES_c	$NESP_p$	$NESP_c$
1	48/50	48/50	48/50	48/50
2	48/50	48/50	46/50	48/50
3	46/50	48/50	46/50	48/50
4	41/50	45/50	40/50	45/50
5	10/50	11/50	9/50	11/50

From the result, we observe that both *NES* and *NES-Prune* can discover most of the relationships. We only count the relationships with the exact same event type, the exact periodical parameters and more than 95% overlap with the ground-truth patterns we plant in, so the criterion is quite demanding.

Compression ratio. We evaluate the quality of event summarization by compression ratio CR with the formula used in [KT09]: $CR(A) = \frac{L(A)}{L(direct)}$, where A is an algorithm, $L(A)$ denote the code length achieved by A , and $L(direct)$ denote the code length of directly encoding the event sequence.

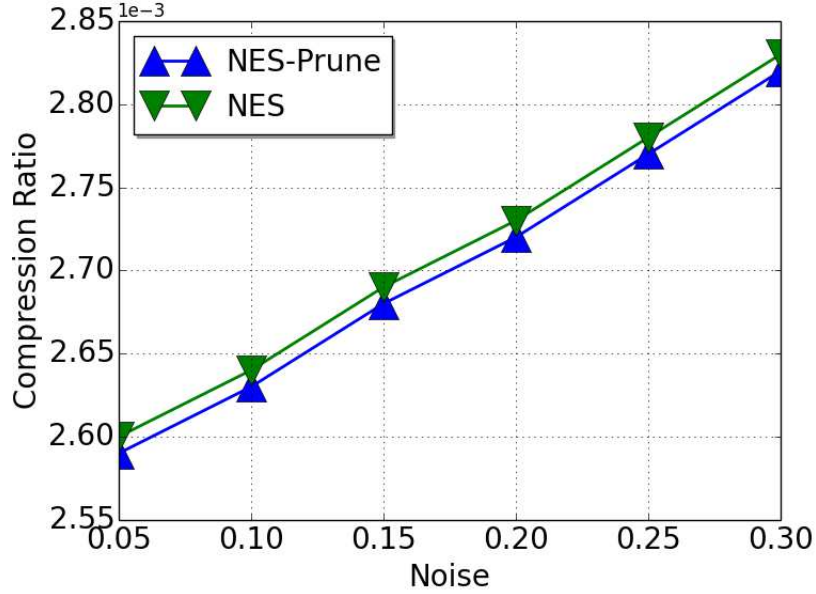


Figure 3.10: CR vs. noise, with parameters

Figure 3.10 shows the compression ratio of our two algorithms as a function of *noiselevel*. For this experiment, we use the same datasets of the accuracy experiments. As expected, *CR* increases as the *noiselevel* increases. It is reasonable because more extra bits are required to describe the noise.

The proportion of planted patterns in the datasets is another factor that affects *CR*. We define the proportion of patterns as $\frac{|Patterns|}{|S|}$, where $|Patterns|$ denotes the number of events belonging to the injected patterns. The experiment is conducted on 6 datasets by fixing $n = 5000, m = 50, l = 100, noiselevel = 0.1$ and the proportion of patterns ranges from 10% to 60% with an increment of 10%. Figure 3.11 shows that *CR* decreases as the pattern proportion increases. The result is straightforward because as the proportion of patterns increases, more events can be well fitted by event patterns, which results in shorter description length for the event sequence.

It should be pointed out that *NES* summarizes an event sequence from different aspects simultaneously using event patterns for the goal of providing natural, interpretable and comprehensive summaries. It is quite possible that some events may

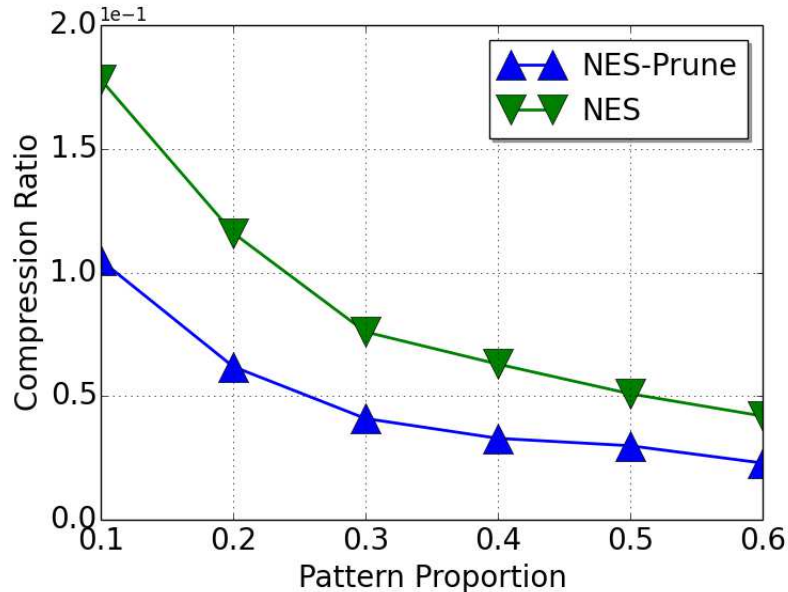


Figure 3.11: CR vs. proportion of pattern

belong to multiple correlation patterns and their coding length might be counted multiple times. So CR might not be the criterion to measure the quantity of summarization for the proposed approach. This phenomenon also explains why NES leads to longer code lengths than $NES-Prune$.

Performance and scalability. We generate 8 datasets by fixing $m = 100, n_c = 0, noiselevel = 0.1$ to evaluate the scalability of our approach. The length of event sequence for the datasets ranges from 2500 to 20000 with an increment of 2500, and n_p is set proportional to the length of sequence. Figure 3.12 shows the scalability results. As expected, $NES-Prune$ runs much faster than NES and it shows better scalability, since *BoundaryPruning* prunes a large number of boundary candidates.

The number of event type m is an important factor that affects the running time, we conduct evaluation on 6 synthetic datasets by fixing $n = 4096, n_c = 0, noise\ level = 0.1$ and m from 4 to 128 with an increment of a power of 2. Figure 3.13 shows that as m increases, the running time of NES decreases but the running time of $NES-Prune$ increases. This is because the running time of NES procedure is $O(|D|^2) = O(m^2|S|^2)$.

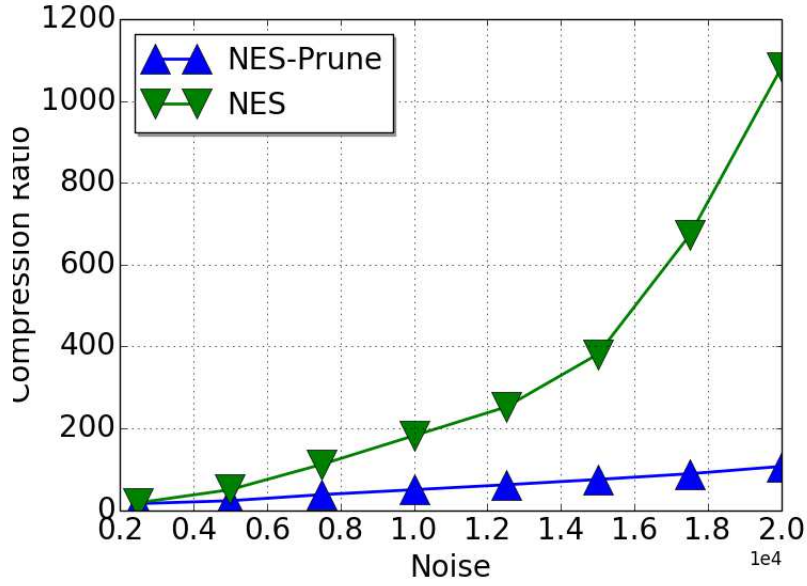


Figure 3.12: Scalability of NES and NES-Prune

Since $S \gg m$ in general cases, for *NES*, the running time is dominated by $|S|$. By fixing n , the average length of S decreases as m increases, therefore the running time of *NES* decreases. For *NES-Prune*, as the input is pruned, the average length of S does not affect running time. Hence, the running time is dominated by m and would increase as m increases,

The proportion of patterns is another important factor that affects the running time. we generate 8 datasets by fixing $n = 5000, m = 50, l = 100, noiselevel = 0.1$ and set the proportion of patterns from 10% to 80%. As Figure 3.14 shows, the increase of the proportion of patterns would decrease the running time of *NES-Prune*, but the running time of *NES* is independent of the proportion of patterns. This is because as the proportion of patterns increases, more boundaries are removed using *BoundaryPruning*. And without *BoundaryPruning*, the size of input for *NES* is independent of the proportion of patterns.

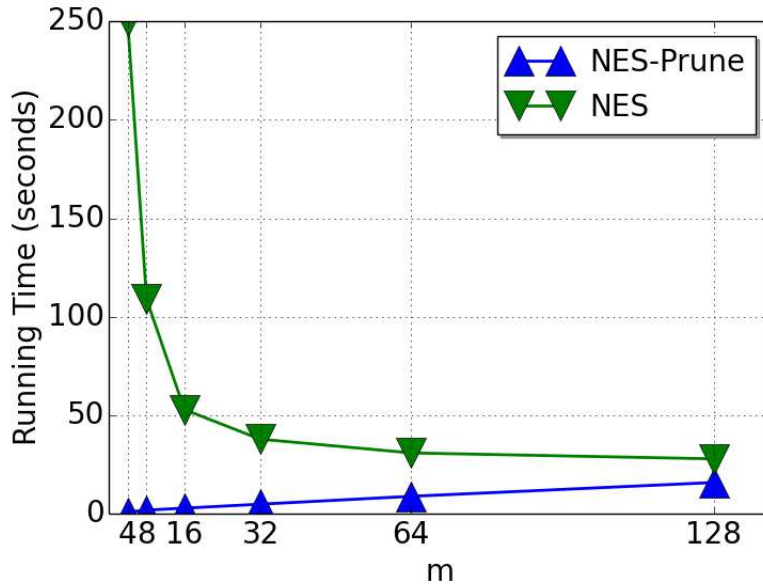


Figure 3.13: Running time vs. number of event type

Experiments on Real Data

In order to explore whether the proposed framework can really help system administrators, we test it with real log datasets recorded in Windows event viewer.

The real datasets consist of **application log**, **security log** and **system log**. The detailed information of the three datasets with their running times and compression ratios are listed in Table 3.4.

Our methods show a significant improvement over previous approaches. The algorithm proposed by Kiernan et al. [KT09] needs more than a thousand seconds to find the optimal summarization solution while our proposed method requires need 10% of the time. Due to inherent difficulty of event summarization, our methods are still not efficient enough. Further improvement in efficiency is needed and this is one of our future works.

Note that our proposed algorithm utilize *MDL* to summarize the event sequence on the aspect of period pattern and correlation pattern, we successfully compressed

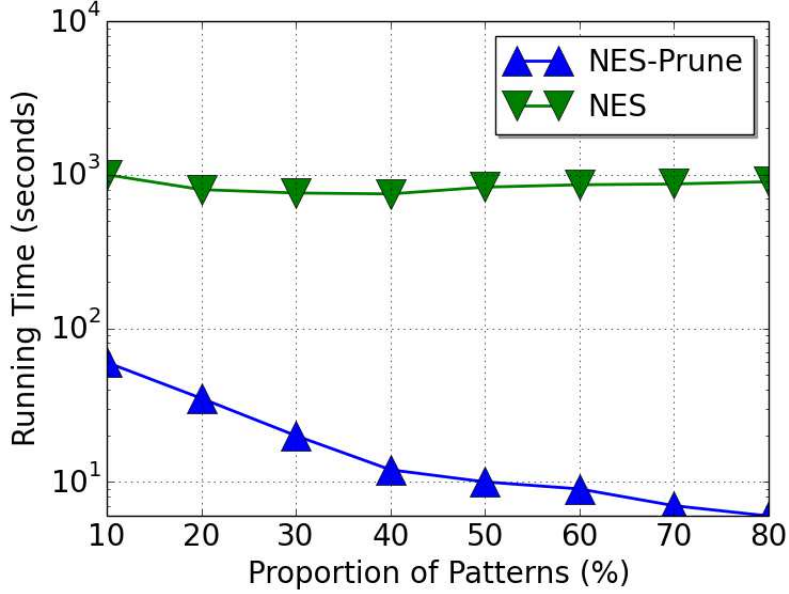


Figure 3.14: Running time vs. Proportion of Patterns

the event sequence to a few percentage of the original amount of data storage space without the losing critical temporal information.

Table 3.4: Experiments with real datasets

	application	security	system
Period	09/10-02/11	11/10-12/10	09/10-02/11
Time range (secs)	12,005,616	2,073,055	12,000,559
Event instances	5634	21850	3935
Event types	100	17	50
Running Time (secs)			
NES	173	3102	108
NES-Prune	22	56	4
Compression Ratio $CR(A)$			
NES	0.0597	0.0312	0.0454
NES-Prune	0.0531	0.0216	0.0464

Our proposed algorithm finds and summarizes a lot of interesting event relationships. For example in system log, we find event $\langle 35, W32Time \rangle$ occurs every 3600 seconds. This event type is a well known periodic event that synchronizes the computer time with the remote server. We also find that there exist correlation

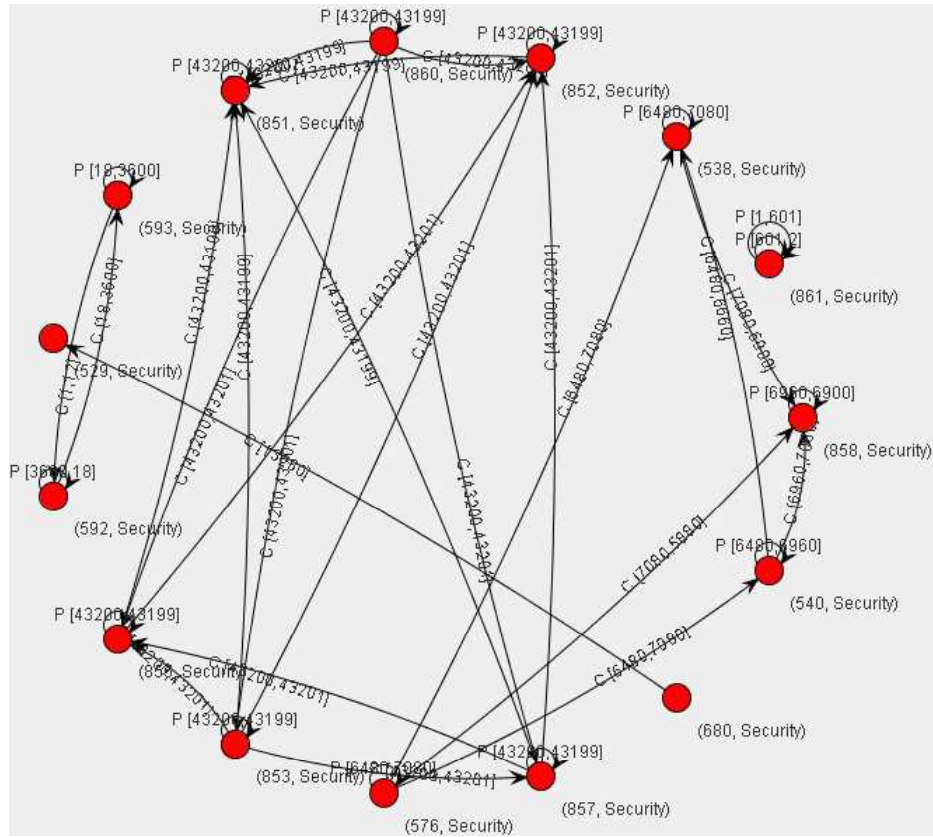


Figure 3.15: ERN for security log

relationships among the event types $\langle 6005, eventlog \rangle$, $\langle 6006, eventlog \rangle$, and $\langle 6009, eventlog \rangle$. The description on EventID.NET¹ verifies that such results are meaningful. Moreover, we find several important relationships from security log and application log. Figure 3.15 shows the whole *ERN* graph for the security log which includes 15 vertices and 36 edges. In this figure, the labels of vertices denote the event types and the labels on edges denote the relationships in form of $C[param1, param2]$, where C or P denotes the type of relationship and $param1, param2$ denote the time interval parameter of the relationship.

¹<http://www.eventid.net>

Table 3.5: Cliques of event relationships and their interpretations

Size	Event Types	Interpretation (After matching clique back to original <i>ERN</i>)
6	(851, security), (860, security) (852, security), (850, security) (853, security), (857, security)	This clique is related to the firewall actions. At first, event 860 (The windows firewall has switched the active policy profile) is triggered, then the other five types are triggered by 860.
4	(576, security) , (540, security) (538, security), (858, security)	This clique is related to the operation of network accessing. At first, event 76 (Special privileges assigned to new logon) is triggered, then event 540 (Logon successful) is triggered, and then event type 538 (User logoff) and 858 (Unknown) happens afterward.
2	(592, security), (593, security)	This clique is related to the creation and destroy of process. Event type 593 (A new process has been created) and type 593 (A process has exited) occur alternatively.
1	(861, security)	This clique is related to the actions of firewall. Event type 861 (The windows firewall has detected an application listening for incoming traffic) appears periodically. This is because the anti-virus scanner installed on this machine continuously monitors the ports.

3.3 Multi-resolution Event Summarization Framework

To facilitate event summarization, an extensible event summarization framework called *META* is proposed. *META* satisfies all the requirements of a generic event summarization framework as mentioned in the motivation (See Chapter 3.1.2). It is able to provide multi-resolution summarization as well as facilitate a set of commonly used summarization tasks. The motivation of presenting *META* is to fill the missing component of the event summarization tasks and make it a complete knowledge discovery process.

META is designed with the following principles: (1) The framework should be flexible enough to handle the daily event summarization scenarios; and (2) The framework should ease the concrete event summarization implementation as much as possible.

Figure 3.16 shows the corresponding workflows of using *META* to handle each of the scenarios mentioned in the motivation (See Chapter 3.1.2). The involved summarization operations include ad-hoc summarization, events storing, recovering, updating, and merging. For each of the operations, the analyst only needs to write and run a short piece of script.

From the technical perspective, *META* leverages a multi-resolution data model called *summarization forest* to efficiently store the events data. *Summarization forest*

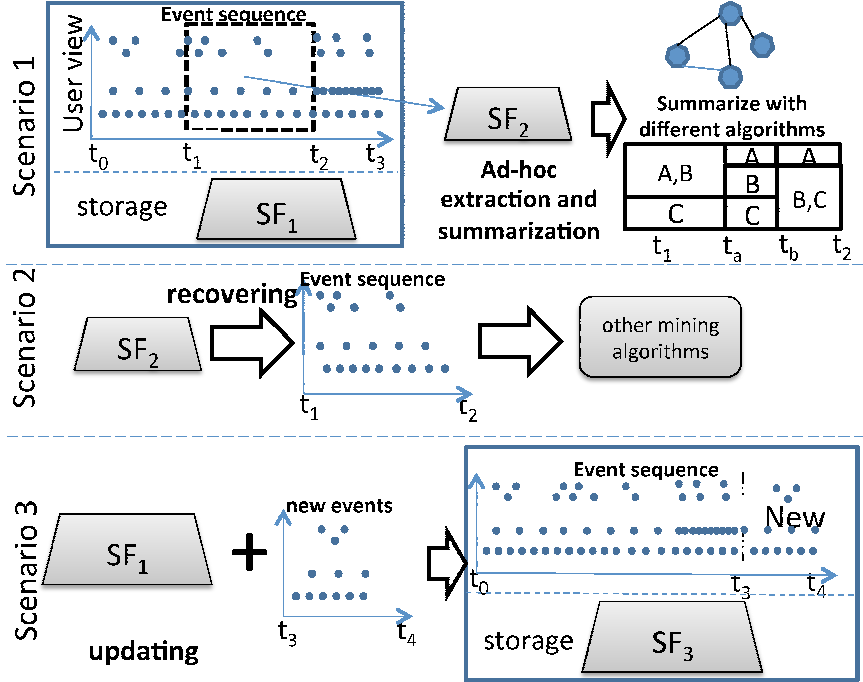


Figure 3.16: Event summarization scenarios and corresponding workflows

is designed to store and represent the events in multi-resolution view with specific precision. On top of *summarization forest*, a query language including a set of basic operations is designed to facilitate the summarization tasks. Moreover, by encapsulating the operations, five commonly used event summarization tasks are introduced to further reduce the work for the event analysts. To better explain how *META* facilitates event summarization, the following chapters would drill down to its details.

3.3.1 The Multi-Resolution Data Model

An event sequence can be represented in the form of event record sequence $D = (\langle t_1, e_1 \rangle, \langle t_2, e_2 \rangle, \dots, \langle t_n, e_n \rangle)$, where t_i is the time when an event occurs and e_i denotes the event instance with an associated ‘type’. Each event instance belongs to one of the m types $\mathcal{E} = \{e_1, \dots, e_m\}$. Note that the ‘type’ is a generic terminology. Any combination of the features of an event can be used as the ‘type’, e.g. the event

category and event name in combination can be used as the event type. In this section, we first describe how to use an *event vector*, an intermediate data structure, to represent the event sequence. Then we introduce *summarization forest* (SF), the data model to store event sequences with multiple resolutions.

Vector Representation of Event Occurrences

Given an event sequence D with m event types and time range $[t_s, t_e]$, we decompose D into m subsequences $D = (D_{e_1}, \dots, D_{e_m})$, each contains the instances of one event type. Afterwards, we convert each D_i into an event vector V_i , where the indexes indicate the time and the values indicate the number of event occurrences. During conversion, we constrain the length of each vector to be $2^l, l \in \mathcal{Z}^+$, where l is the smallest value that satisfies $2^l \geq t_e - t_s$. In the vector, the first $t_e - t_s$ entries would record the actual occurrences of the event instances, and the remaining entries are filled with 0's. Example 4 provides a simple illustration on how we convert the event sequence.

Example 4. *The left figure in Figure 3.17 gives an event sequence containing 3 event types within time range $[t_1, t_{12}]$. The right figure shows the conversion result of the given event sequence. Note that the original event sequence is decomposed into 3 subsequences. Each subsequence representing one event type is converted to a vector with length 16. The numbers in bold indicate the actual occurrences of the events, and the remaining numbers are filled with 0's.*

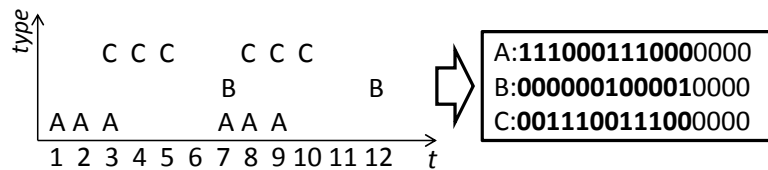


Figure 3.17: Convert the original event sequence to the vectors

Vectors intuitively describe the occurrences of events, but this kind of representation is neither storage efficient (as it requires $O(|\mathcal{E}|n)$) nor analysis efficient (as it does not support multi-resolution analysis). Firstly, directly storing these vectors requires $O(|\mathcal{E}|n)$ space, which is even more than the original event sequence. Secondly, the format of vector is not suitable for event summarization and multi-resolution analysis. It cannot enable analysts to efficiently retrieve and view the occurrences of events at an arbitrary granularity. To facilitate the storage and analysis, we propose *summarization tree* to model the event occurrences of a single type. Furthermore, we propose *summarization forest* to model the event occurrences of the whole event log.

Summarization Tree

The summarization tree is used to store the event occurrences for a single event type. It is capable of providing both frequency and locality of occurrences simultaneously. Moreover, it satisfies the multi-resolution analysis (MRA) [Mal89] requirements by representing the event occurrences with various subspaces. This property enables the analysts to choose a proper subspace to view the data at a corresponding granularity. The summarization tree is formally defined below.

Definition 3.3.1. *A **summarization tree** (ST) is a balanced tree where all nodes store the temporal information about the occurrences of events. The tree has the following properties:*

1. *Each summarization tree has two types of nodes: summary node and description nodes.*
2. *The root is a summary node, and it has only one child. The root stores the total occurrences of the events throughout the event sequence.*
3. *All the other nodes are description nodes. They either have two children or have no child. These nodes store the frequency difference between adjacent chunks*

(the frequency of the first chunk subtracted by the frequency of its following chunk) of sequence described by lower level nodes.

4. The height of the summarization tree is the number of levels of the description tree. The height of a node in tree is counted from bottom to top, starting from 0. The nodes at height i store the frequency differences that can be used to obtain the temporal information of granularity i .¹

Considering event type A in Example 4, Figure 3.18 shows its vector and the corresponding summarization tree. As illustrated, the summarization tree stores the sum of the occurrences frequency (6 occurrences) at the root node, and the frequency differences (within the dashed box) in the description nodes at various granularities. Note that at the same level of the tree, the description nodes store the differences between adjacent sequence chunks at the same granularity. The larger the depth, the more detailed differences they store. For example, at granularity 1, every two adjacent time slots in the original event sequence are grouped into one chunk, and the grouped event sequence is ‘21021000’. Correspondingly, in the summarization tree, the frequency differences of each adjacent time slot $(0, -1, 0, 0, -1, 0, 0, 0)$ are recorded at the leaf level. Similarly, the frequency differences at various granularities are recorded in the description nodes at the corresponding levels.

It is clear that the space complexity of the summarization tree is $O(|T|)$, where $|T| = n$ and n is the length of the vector. From the storage perspective, directly storing the tree has no benefits for space saving. Basically, there are two ways to reduce the space complexity of summarization tree: *detail pruning* and *sparsity storage*.

¹According to the property of MRA, the time precision of nodes decreases exponentially as the index of their granularity increases. For example, if the precision of the original data is 1 second, the corresponding tree nodes at granularity i represent the data every 2^i seconds.

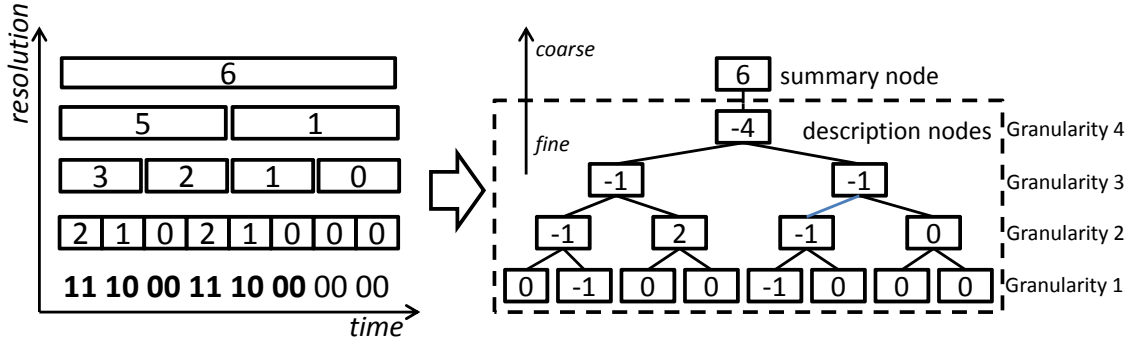


Figure 3.18: Relationship between vector and ST

Detail Pruning In practice, analysts only care about the high-level overview of the event occurrences. Consequently, there is no need to store all the details of the event sequences. As the summarization tree describes the event occurrences in a top-down manner — a coarse-to-fine strategy, we can save the storage by removing the lower levels of the description nodes. The pruned tree still contains enough details for analysis, and an analyst who analyzes a long-term event log would not care about the event occurrences at the second precision. Due to the hierarchical structure of the tree, we can reduce the storage space exponentially. Lemma 3.3.2 shows how much space can be reduced through pruning. For example, the original tree has a height of 14 levels and 8192 (or 2^{13}) nodes. If we prune the tree by removing the last 6 levels, the size of tree will become $\frac{1}{2^6}|T| = 128$, which is only about 1.5% of the original size. The pruned tree is still able to describe the event occurrences with 1-minute granularity.

Lemma 3.3.2. *Suppose the height of summarization tree is H , if only the nodes with a height larger than or equal to k are kept, the size of the pruned tree is 2^{H-k} .*

Proof. According to the property 3 of the definition of summarization tree, besides the summarization node, the summarization tree is a perfect binary tree. If only the nodes with height larger than or equal to k are kept, the size of remaining nodes in

perfect binary tree part is $\sum_{i=k}^{H-1} 2^{H-1-i} = 2^{H-k} - 1$. Therefore, the total size of the summarization tree after pruning is 2^{H-k} . \square

Sparsity Storage Another way to reduce the space is to store the non-empty nodes of the tree only. The majority of the event types rarely appear in the event sequence. In this case, the corresponding vector will be dominated with 0's. Accordingly, the transformed summarization tree will also contain many 0's. For example, events with type X only occur twice throughout a 2-hour (7200 second) event sequence. The first occurrence is the first second, and the second occurrence is the second second. The number of nodes in the corresponding summarization tree is 8192, but there are only 28 non-zero nodes. Lemma 3.3.3 provides a lower bound on how many zero nodes exist in a summarization tree.

Lemma 3.3.3. *Suppose the occurrence proportion (the probability of occurrences at any time) of event type X is $r = \frac{\#X}{n}$, where n is the length of vector that stores the event occurrences, the proportion of zero nodes at height h is $p_h = \max(1 - 2^{h+1}r, 0)$ for the corresponding summarization tree.*

Proof. We calculate the number of zero nodes from the bottom level to the top level. It is trivial to know that besides the root level, the number of nodes at height h is $n_h = \frac{|T|}{2^{h+1}}$. For each level, the number of zero nodes z_h equals to the number of nodes n_h minus the number of non-zero nodes u_h .

We start with $h = 0$ (the leaf level). In the worst case, the event occurrences are uniformly distributed along the time-line. There are two cases according to r :

1. $0 \leq r < \frac{1}{2}$. The event occurs in less than half of the time slots. In such condition, $u_0 = \min(r|T|, n_0)$, and $z_0 = n_0 - u_0 = \frac{|T|}{2} - r|T|$. So $p_0 = \frac{z_0}{n_0} = 1 - 2r$.

2. $\frac{1}{2} \leq r \leq 1$. The number of zero nodes at the leaf level can be 0. Since occurrences are uniformly distributed, it is possible that the event appears at least once in every two continuous time slots. In this case, $p_0 = 0$.

Therefore, the lower bound probability of the zero nodes at the leaf level is $p_0 = \max(1 - 2r, 0)$. When $h = 1$, in the worst case, the occurrences of non-zero nodes at the leaf level are still uniformly distributed, so $u_1 = \min(u_0, n_1)$. Therefore, $z_1 = n_1 - u_1 = \max(n_1 - u_0, 0)$ and $p_1 = \max(1 - 2^2r, 0)$. When $h > 1$, if the occurrences of non-zero nodes at a lower level is still uniformly distributed, the number of zero nodes $u_h = \min(u_{h-1}, n_h)$. Similar to the case of $h = 1$, $z_h = n_h - u_h$, and $p_h = \max(1 - 2^{h+1}r, 0)$. \square

Based on Lemma 3.3.2 and 3.3.3, we further show the space complexity of a summarization tree in Theorem 3.3.4.

Theorem 3.3.4. *The space complexity of a summarization tree with granularity k is $O(\frac{|T|}{2^k} - \sum_{i=k}^H \max(\frac{|T|}{2^i} - 2^{i+1}r, 0))$, where $|T|$ is the length of the vector, H is the height of the summarization tree, and r is the occurrence proportion as described in Lemma 3.3.3.*

Proof. The proof is based on Lemma 3.3.2 and Lemma 3.3.3. The number of nodes with the height (granularity) larger than or equal to k is $2^{H-k} = \frac{|T|}{2^k}$ according to Lemma 3.3.2. For each level $h \geq k$, the number of zero nodes is $n_h p = \max(\frac{|T|}{2^h} - 2^{h+1}r, 0)$, and the sum of all nodes with height larger than or equal to k is $\sum_{i=k}^H \frac{|T|}{2^i} - 2^{h+1}r$. Therefore, the number of non-zero nodes in the summarization tree is $\frac{|T|}{2^k} - \sum_{i=k}^H \max(\frac{|T|}{2^i} - 2^{i+1}r, 0)$. \square

It is true that the second term will become 0 when r is sufficiently large. However, based on the empirical study, most of the event types occur rarely, and therefore $0 < r \ll 1$.

Summarization Forest

Summarization forest is a data model which contains all the summarization trees. In one forest, there are $|\mathcal{E}|$ summarization trees. Each stores the events of one event type. Besides trees, the summarization forest also stores the necessary meta-data. The summarization forest is formally defined in Definition 3.3.5.

Definition 3.3.5. A summarization forest (*SF*) is a 6-tuple $\mathcal{F} = \langle \mathcal{E}, \mathcal{T}, t_s, t_e, l, r \rangle$, where:

1. \mathcal{E} denotes the set of the event types in the event sequence.
2. \mathcal{T} denotes the set of summarization trees.
3. t_s and t_e denote the start timestamp and end timestamp of the event sequence represented by \mathcal{F} .
4. l denotes the full size of each *ST*, including the zero and non-zero nodes. All the trees have the same full size.
5. r denotes the resolution of each *ST*. All the trees are in the same resolution.

Note that since the summarization trees are stored in sparsity style, the actual number of nodes that are stored for each tree can be different and should be much less than the full size. Given a summarization forest, we can recover the original event sequences.

3.3.2 Basic Operations

In this section, we propose a set of basic operators which are built on top of the data model we proposed. These operators form the summarization language, which is the foundation of the event summarization tasks presented in our framework. The motivation of proposing a summarization language is to make the event summarization

flexible and allow the experienced analysts to define the ad-hoc summarization tasks to meet the potential new needs.

The basic operators are categorized into two families: the *data transformation operators* and the *data query operators*. The operators of the first family focus on transforming data from one type to another, and they are not directly used for summarization work. The operators of the second family focus on retrieving/manipulating data in read-only way, and they provide the flexibility of generating the summarization. To make the notations easy to follow, we list all the symbols of all these operations in Table 3.6. We will introduce their meanings later in this section.

Data Transformation Operators

The data transformation operators includes *vectorize*, *unvectorize*, *encode*, *decode*, *prune*, and *concatenate*. Their functionalities are listed as follows:

- **Vectorize** and **Unvectorize**: *Vectorize* is used to convert the single event type subsequence D_i into a vector V_i while *unvectorize* does the reverse work. Both of them are unary, and represented by symbol \circ and \bullet , respectively. Semantically, these two operators are complementary operators, i.e. $D_i = \bullet(\circ(D_i))$ and $V_i = \circ(\bullet(V_i))$.
- **Encode** and **Decode**: *Encode* is used to convert the vector V_i into a summarization tree T_i while *decode* does the reverse work. Similar to *vectorize/unvectorize*, *Encode* and *decode* are complementary operators and both of them are unary. We use symbol \triangleleft and \triangleright to denote them respectively.
- **Prune**: The operator *Prune* is unary, and it conducts on the summarization tree. It is used to remove the most detailed information of the events by pruning the leaves of a summarization tree. Note that this operator is irrecoverable.

Operation	Symbol	Description
<i>Vectorize</i>	$\circ(D_i)$	Vectorize the subsequence D_i .
<i>Unvectorize</i>	$\bullet(V_i)$	Unvectorize the vector V_i .
<i>Encode</i>	$\triangleleft(V_i)$	Encode V_i into a summarization tree T_i .
<i>Decode</i>	$\triangleright(T_i)$	Decode T_i back to vector V_i .
<i>Prune</i>	$\ominus(T_i)$	Prune the most detailed information of T_i .
<i>Concatenate</i>	$\mathcal{F}_1 \uplus \mathcal{F}_2$	Concatenate two SF \mathcal{F}_1 and \mathcal{F}_2 .
<i>Project</i>	$\Pi_{e_{(1)}, \dots, e_{(k)}}(\mathcal{F})$	Extract events of types $e_{(1)}, \dots, e_{(k)}$ from \mathcal{F} .
<i>Select</i>	$\sigma_{[t_1, t_2]}(\mathcal{F})$	Pick the events occurs between time $[t_1, t_2]$.
<i>Zoom</i>	$\tau_i(\mathcal{F})$	Aggregate the events with granularity u .
<i>Describe</i>	Υ_{name}	Use algorithm <i>name</i> for event summarization.

Table 3.6: Notations of basic operations

Once it is used, the target summarization tree will permanently lose the removed level. We use \ominus to denote this operator.

- **Concatenate:** The operator *concatenate* is a binary operator. It combines two SFs into a big one and also updates the meta-data. We use \uplus to denote this operation. Note that only the SFs with the same resolution can be concatenated.

Data Query Operators

The data query operators include *select*, *project*, *zoom*, and *describe*. They all take the *summarization forest* \mathcal{F} as the input. The data query operators are similar to the *Data Manipulation Language (DML)* in SQL, which provides query flexibility to users.

Their functionalities are listed as follows:

- **Project:** The operator *project* is similar to the ‘projection’ in relational algebra. It is a unary operator written as $\Pi_{e_{(1)}, e_{(2)}, \dots, e_{(k)}}(\mathcal{F})$. The operation is defined as picking the summarization trees whose event types are in the subset of $\{e_{(1)}, \dots, e_{(k)}\} \subseteq \mathcal{E}$.
- **Select:** The operator *select* is similar to the ‘selection’ in relational algebra. It is a unary operator written as $\sigma_{[t_1, t_2]}(\mathcal{F})$.

- **Zoom:** The operator *zoom* is used to control the resolution of the data. It is a unary operator written as $\tau_u(\mathcal{F})$, where u is the assigned resolution, the larger, the coarser.
- **Describe:** The *describe* operator indicates which algorithm is used to summarize the events. Its implementation depends on the concrete algorithm and all the previous event summarization papers can be regarded as proposing a concrete describe operator. For example, [JPL11] summarize the events with periodic and inter-arrival relationships. The *describe* operation is written as $\Upsilon_{name}(\mathcal{F})$, where *name* is the name of summarization algorithm used for describing the events. If necessary, the analysts can implement their own *describe* algorithms that follow the specification of our framework. In our implementation, the time complexity of all these operators are lower than $O(|\mathcal{E}||T| \log |T|) = O(|\mathcal{E}|n \log n)$.

Table 3.7 illustrates the time complexity of all the operations and provides short comments about how they are implemented. The most time-consuming operations are encode/decode operations. This is because these two operations are based on wavelet transformation whose lower bound time complexity is proved to be $O(n \log n)$, the inherent characteristic makes it impossible to have more efficient implementation from the algorithmic perspective. In practice, the operations perform much faster than the theoretical time complexity. This is because SF is stored in the sparse format, making the actual size of summarization trees to be far less than $|T|$. In SF, the summarization trees of different event types are independent, therefore, the encode/decode operations can be conducted in parallel.

As for the *describe* operation, the time complexity is not given since its implementation is depend on the summarization algorithm designer. *META* provides a well designed interface and take the algorithm designer’s implementation as a plug-

in. This design strategy is similar to the design of SQL, where the interface of the query function is opened to the function designer.

Operator	Time Complexity	Comment
Vectorize & Unvectorize	$O(\mathcal{E} T)$	Scan the event log once and transform them into sparse vector, and vice versa.
Encode & Decode	$O(\mathcal{E} T \log T)$	Leverage the wavelet transformation and inverse wavelet transformation to conduct the encode/decode operation.
Prune	$O(\log \mathcal{E} T)$	Directly delete the nodes of trees in SF located at lower resolution. The threshold index can be calculated in constant time.
Concatenate	$O(\mathcal{E} T)$	For each tree in first SF, find its counterpart in the second SF; and sum the value of the node in first tree with index $i + offset$ and the value of the node in second tree with index i , where $offset$ is the start time difference between the two SFs.
Project	$O(\mathcal{E})$	Pick the selected trees from the SF by checking the event type each tree represents.
Select	$O(\mathcal{E} T \log T)$	Extract the data which describes the events during $[t_1, t_2]$ from the original summarization trees; and build a new tree from the extracted data.
Zoom	$O(\mathcal{E} T ^{\frac{\log T }{u}})$	Similar to Prune, but avoid destroying the original data by creating a copy of the needed tree nodes.

Table 3.7: Time complexity of summarization operations

3.3.3 Event Summarization Tasks

Considering the requirements of the analysts discussed in Introduction, we introduce five commonly used event summarization tasks: *summarization*, *storing*, *recovering*, *merging*, and *updating*, using the previously defined basic operators as the building blocks. The intention here is to demonstrate the expressive capability of the basic operators, instead of giving a thorough coverage of all the possible tasks.

Summarization Task

Summarization task is the core of event summarization, and all prior works about event summarization focus on this problem. Based on the defined basic operators, analysts can summarize the events in a flexible way. In our framework, any summarization task can be described by the following expression:

$$\Upsilon_{name}(\sigma_{[t_1, t_2]}^* \tau_u^* \Pi_{E \in \mathcal{P}(\mathcal{E})}^*(\mathcal{F})).$$

The symbol $*$ denotes conducting the operation 0 or more times. With the combination of operators, the analysts are able to summarize **any** subset of events in **any** resolution during **any** time range with **any** summarization algorithm.

One thing should be noted is that the order of the operators can be changed, but the summarization results of different orders are not guaranteed to be the same. For example, commonly used implementations [JPL11,KT09] of the *describe* operator are based on the minimum description length principle. Such implementations aims to find models that describe the events with least information. Therefore, the results of $\Upsilon_{name}(\tau_u(\mathcal{F}))$ and $\tau_u(\Upsilon_{name}(\mathcal{F}))$ are possibly different.

Storing Task

Storing is an important task. Converting the raw event log time after time is time-consuming with low management efficiency. This task enables the analysts to convert the events into a uniform data mode only once and reuse it afterwards. The store task can be written as:

$$\mathcal{F} = \bigcup_{e_i \in \mathcal{E}_I} \Theta^*(\triangleleft(\circ(D_i))),$$

where \mathcal{E}_I denotes the set of event types that the analysts are interested in, and \bigcup denotes putting all the trees together to form the SF. The analysts are able to pick **any** time resolution and **any** subset of all the event types for storage.

Recovering Task

Recovering task is the link between the event summarization and other data mining tasks. After finding the interesting piece of event logs via the summarization results, the analysts should be able to transform the selected portion of SF back to its original events, so they can use other data mining techniques for further analysis. The recover task can be expressed as:

$$\bullet(\triangleright(\sigma_{[t_1, t_2]}^*(\tau_u(\Pi_{E \in \mathcal{E}_I}^*(\mathcal{F}))))).$$

This expression shows that the analysts can selectively recover the piece of events with **any** subset of event types, at **any** time range and **any** time resolution.

Merging and Updating Tasks

Both merging and updating tasks focus on the maintenance of stored SF, but their motivations are different.

The merging task is conducted when the analysts obtain the SFs with disjoint time periods and want to archive them altogether. Suppose \mathcal{F}_1 and \mathcal{F}_2 denote two SFs, where \mathcal{F}_2 contains more details (contains lower resolution level). The merging task can be expressed as:

$$\mathcal{F}_{new} = \mathcal{F}_1 \uplus \ominus^*(\mathcal{F}_2).$$

As shown in the above expression, when we merge two summarization trees with different resolutions, the SF with higher granularity would be pruned to meet the SF with lower granularity. Then these two SFs would be merged with the *concatenate* operation.

Updating task is conducted when the analysts want to update the existing SF with a new piece of event log. It can be expressed by basic operators as follows:

$$\mathcal{F}_{new} = \mathcal{F} \uplus \left(\bigcup_{e_i \in \mathcal{E}_I} \ominus^*(\triangleleft(\circ(D_i))) \right),$$

where the operand of \cup is similar to the operand of \cup in storing task. Firstly, the new set of subsequence D_i will be vectorized and then encoded into a SF \mathcal{F} . Then the new SF would be merged into the old SF same as the *merge* task.

3.3.4 Experimental Evaluation

We conduct a series of experiments to evaluate our proposed framework. In this section, we do not focus on demonstrating the meaningfulness or correctness of the summarization results, since it should be the work of the concrete summarization algorithm designers. Instead, the main goal of the evaluation is to explore the efficiency and the effectiveness of the proposed framework, and to show how *META* makes the summarization more flexible and convenient. More concretely, our experiments aim to answer the following questions: (1) What is the cost to store the events in the form of SF? (2) How efficient is it to retrieve and convert the data from the SF? (3) How effective and flexible can our framework support the event summarization? and (4) What about the performance of the updating and merging tasks?

In addition to the evaluation of *META*, we also give a case study to show how *META* facilitates analysts to conduct event summarization tasks. As a showcase, we leverage the algorithm proposed in [JPL11] as the summarization algorithm, which summarizes the events from the perspective of inter-arrival temporal relationship.

Storage Cost

To evaluate the storage cost of a *SF*, we do experiments by using several real event logs across different OS platforms and domains. These event logs are collected from customer's servers by IBM service department and the details of these logs are listed in Table 3.8 (Available at <http://share.olidu.com/events/>). These datasets are different

in the aspect of time range, event occurrences, occurrences frequency, distinct event types, and log record styles.

Name	Domain	Time Units	#Types
secure-secure	Security	534,898	14
nokia-netview	Network	99,118,589	15
system-win	System	41,113,840	64
security-win	Security	5,579,292	35
application-win	Application	6,980,559	61

Table 3.8: Features of real datasets

Table 3.9 illustrates the occurrence proportion of the events in the real world datasets used in the experiments. We record the maximum, average, and minimum occurrence proportion of the event types in each dataset. Among all the datasets, the most frequent event type has the occurrence proportion 0.022, indicating the event occurs only 22 out of every 1000 time slots throughout the time range of the event sequences. The data in this table demonstrates that no event type occurs all the time (the occurrence proportion $r \ll 1$) in real world situation. Therefore, the second term of the O -notation in Theorem 3.3.4 is comparable to the first term, and it makes the theorem meaningful.

	Maximum	Average	Minimum
secure-linux	0.005	0.001	3.739×10^{-6}
nokia-netview	2.185×10^{-4}	4.064×10^{-5}	1.009×10^{-8}
system-win	4.886×10^{-4}	1.413×10^{-4}	2.432×10^{-8}
security-win	0.022	9.381×10^{-4}	1.792×10^{-7}
application-win	0.003	5.787×10^{-5}	1.432×10^{-7}

Table 3.9: Occurrence proportion in real datasets

In order to measure storage cost, we store the SFs as binary files using object serialization technology. We use the *compression ratio* (CR) to quantify the ratio of SF files comparing with the original log file, i.e., $CR = \frac{size(file)}{size(original_file)}$. To further save the storage space, we leverage DEFLATE algorithm [Dav07] to compress the serialized SF. DEFLATE is a widely used data compression algorithm that is a combination of

Lempel-Ziv (LZ77) [ZL78] and Huffman coding, and it has been adopted in several well known software such as 7Zip and WINRAR. Figure 3.19 shows the compression ratio of all the datasets. It can be observed that all the stored SFs cost less storage space comparing with the original logs ($CR < 1$). Moreover, after compressed by DEFLATE, even the worst compressed SF costs only 32.4% of the space of the original log file. This fact shows that storing the logs as SFs can save the disk space.

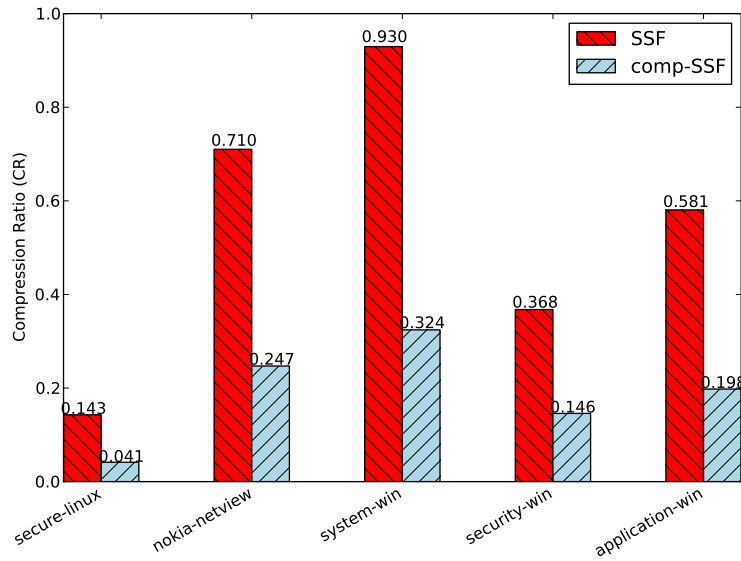


Figure 3.19: Compression ratio of SF and compressed SF

The storage cost can be further reduced if the low level details are pruned. Figure 3.20 shows the compression ratio of each SF in different 7 resolutions without compression. Note that the values in the first row of x-label indicate the level of resolution we store the SFs, and the values in the second row indicate the corresponding approximate time resolution. As depicted in this figure, when the resolution is 13 (hourly resolution), even the most costly SF uses only 5% of space compared with the corresponding original file.

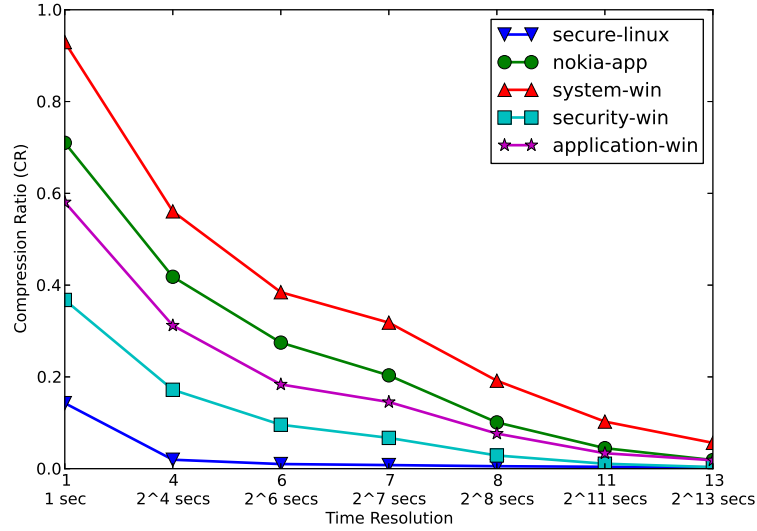


Figure 3.20: Compression ratio of SF in different resolution

Efficiency Evaluation

The efficiency evaluation is conducted in 3-fold. Firstly, we evaluate the performance of the summarization task by exploring different data query operators permutations. Moreover, we measure the performance of storing and recovering tasks to evaluate the time overhead of conducting event summarization within our framework. Finally, we investigate the performance of merging and updating tasks to evaluate the maintenance overhead.

We generate 15 synthetic datasets and investigate the performance of our framework on different datasets by changing 3 properties as listed in Table 3.10. The advantage of using synthetic datasets is that we can evaluate the performance of our framework with different properties systematically. Since we only investigate the efficiency, the occurrences of events are randomly generated.

Performance of Summarization Task In this section, we evaluate the performance of all data query operators except *describe*. The reason is that the performance of *describe* depends on concrete summarization algorithms.

property	values	description
#types	20-100 step 20	The number of event types.
#events	60k-140k step 20k	The number of event occurrences.
#ts	10m-50m step 10m	The time slots in the time range.

Table 3.10: Properties of synthetic datasets

Similar to *Data Manipulate Language* in SQL, the performance of the query varies with different operator permutations. To investigate how the order affects the query performance, we pick three sets of synthetic datasets to evaluate the time cost of different *project*, *select* and *zoom* permutations. In each set, we fix two properties and changes the third one. For *project*, we pick 10% of the event types from the SF. For *select*, we pick 10% of the time range, and zoom out the SF for one resolution. Table 3.11 shows the running time of all the 6 different permutations in 3 sets of experiments. Examining the experiment results in different perspectives, we can obtain following observations:

1. **Different operators have different time costs.** Table 3.11 shows that *select* is the most time-consuming and *project* is the most time-efficient. In our experiments, *select* is $10^2 \sim 10^4$ times slower than *zoom* and *project*. The reason is that by taking advantage of the SF, *zoom* only needs to remove all the leaves from trees in $O(\log |T|)$ time and *project* only needs to remove the useless trees in $O(|\mathcal{E}|)$ time. However, *select* is more complicated than the other two operators. It builds a new SF by extracting events satisfying the *select* parameters from the old SF, which takes $O(|T| \log |T|)$ time. For example, if the analysts intend to view only half of the time range of the event sequence, the other half of the data would be useless and the *select* operation should build a new forest from the scratch based on the remaining events.
2. **Query performance varies drastically according to different operator orders.** The experiment results show that the fastest query costs only 3% the

Order Dataset	select-project-zoom			select-zoom-project			project-select-zoom		
	select	zoom	proj	select	zoom	proj	select	zoom	proj
100-60k-50m	72.98	0.006	0.001	84.55	0.006	0.001	84.52	0.005	0.001
100-80k-50m	71.63	0.011	0.001	82.65	0.007	0.001	84.73	0.007	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001
100-120k-50m	84.28	0.008	0.001	85.11	0.009	0.001	84.71	0.009	0.001
100-140k-50m	82.16	0.010	0.001	84.73	0.010	0.001	85.13	0.010	0.001
20-100k-50m	16.32	0.005	0.001	16.21	0.005	0.001	15.48	0.005	0.001
40-100k-50m	33.43	0.006	0.001	30.85	0.007	0.001	31.13	0.007	0.001
60-100k-50m	48.37	0.008	0.001	46.31	0.007	0.001	46.53	0.008	0.001
80-100k-50m	64.34	0.008	0.001	62.10	0.008	0.001	62.09	0.007	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001
100-100k-10m	18.36	0.007	0.001	18.97	0.006	0.001	18.13	0.006	0.001
100-100k-20m	36.83	0.006	0.001	36.56	0.007	0.001	36.66	0.007	0.001
100-100k-30m	39.86	0.008	0.001	39.44	0.008	0.001	39.98	0.008	0.001
100-100k-40m	77.29	0.009	0.001	76.71	0.008	0.001	74.61	0.008	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001
Order Dataset	project-zoom-select			zoom-project-select			zoom-select-project		
	select	zoom	proj	select	zoom	proj	select	zoom	proj
100-60k-50m	2.39	0.02	0.001	2.45	0.02	0.001	2.40	0.02	0.001
100-80k-50m	2.44	0.02	0.001	2.53	0.02	0.001	2.39	0.02	0.001
100-100k-50m	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001
100-120k-50m	2.51	0.03	0.001	2.53	0.03	0.001	2.46	0.03	0.001
100-140k-50m	2.52	0.04	0.001	2.57	0.04	0.001	2.57	0.03	0.001
20-100k-50m	0.55	0.02	0.001	0.55	0.02	0.001	0.51	0.24	0.001
40-100k-50m	0.99	0.02	0.001	0.99	0.02	0.001	0.99	0.02	0.001
60-100k-50m	1.46	0.02	0.001	1.46	0.02	0.001	1.44	0.03	0.001
80-100k-50m	2.12	0.03	0.001	2.04	0.03	0.001	2.04	0.03	0.001
100-100k-50m	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001
100-100k-10m	0.62	0.04	0.001	6.23	0.02	0.001	0.59	0.03	0.001
100-100k-20m	1.16	0.05	0.001	1.19	0.03	0.001	1.29	0.03	0.001
100-100k-30m	1.17	0.03	0.001	1.30	0.03	0.001	1.30	0.03	0.001
100-100k-40m	2.22	0.03	0.001	2.47	0.03	0.001	2.37	0.03	0.001
100-100k-50m	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001

Table 3.11: Running time composition of different query orders (time unit: second)

time of the slowest query on the same dataset. As mentioned before, *select* is the slowest operator. The more data it processes, the slower the execution would be. Therefore, the later the *select* operation is conducted, the shorter the query execution time would be.

3. **Query performance is insensitive to #events.** According to the experiment results conducted on the datasets with the same #types and #ts (1st group), the query performance appears to be stable when #events increases.

On the contrary, the experiment results on the datasets with the same $\#events$ and $\#ts$ (2nd group) show that the query time varies linearly. Also, the results are similar for the datasets with the same $\#types$ and $\#events$ but with different $\#ts$ (3rd group).

Based on the above observations, to avoid unnecessary time cost, a good query statement should postpone *select* as much as possible. In our prototype, we conduct simple query optimization by reordering the operators.

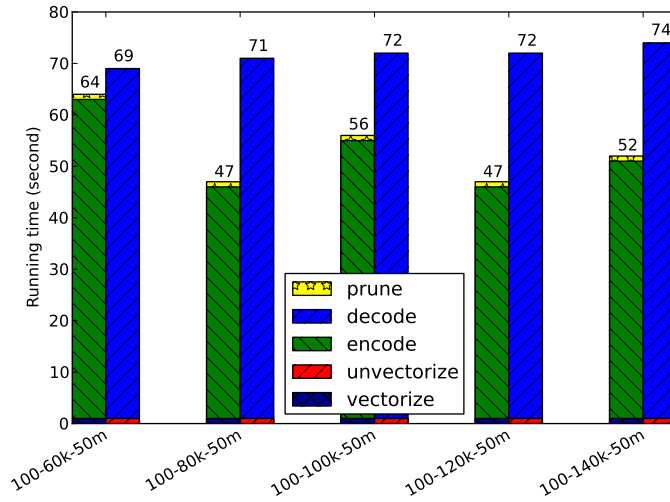


Figure 3.21: Datasets with different $\#events$

Framework Time Overhead The tasks of storing and recovering are not directly related to event summarization, and they are considered as overhead for summarization. We conduct experiments on the same sets of datasets that are used in Section 3.3.4. For each datasets sets, we investigate the time cost of storing and recovering by revealing the running time of involved operators: *vectorize*, *encode*, *prune* for storing task and *decode*, *unvectorize* for recovering task.

Figures 3.21, 3.22, and 3.23 show the experiment results of the time overhead, where the first bar of each dataset indicates the overhead of storing task and the

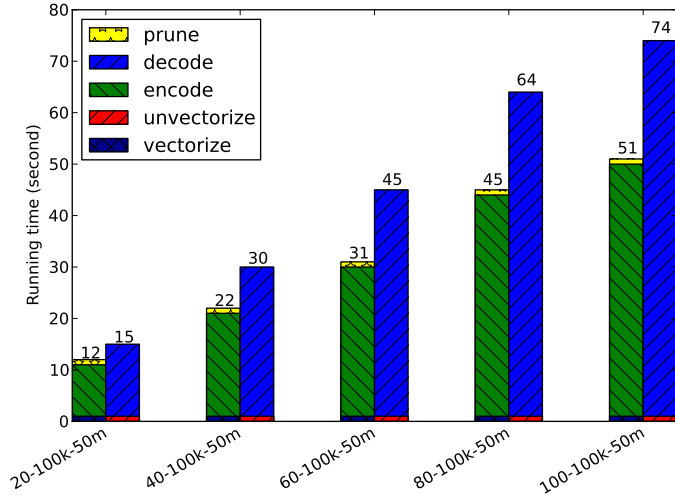


Figure 3.22: Datasets with different #types

second bar indicates the overhead of recovering task. From the experiment results, we obtain two following observations. Firstly, all the experiments cost tens of seconds to finish the tasks. Due to the rare usage of these two tasks, the overhead is acceptable. Secondly, the time overhead of these two tasks are insensitive to #events but sensitive to #types and #ts. As we drill down to the operator level, we find that most of the increased running time comes from the *encode* operator in storing task and *decode* operator in recovering task. In our implementation, both of these two operations have the same time complexity $O(|\mathcal{E}||T| \log |T|)$. The running time would increase if either $|\mathcal{E}|$ or $|T|$ increases. Also, the distribution of event occurrences is another factor to affect the running time.

Performance of SF Maintenance In this section, we investigate the performance of maintenance tasks on two aspects: how the characteristics of events and how the resolution of data affects the performance. Similar to previous experiments, the performances of both tasks using the same three groups of datasets are evaluated. For each group, the first dataset is converted into a SF, and other datasets are incre-

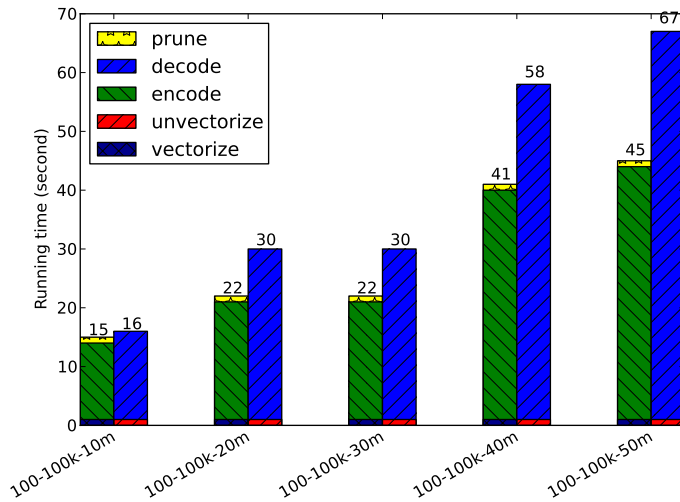


Figure 3.23: Datasets with different #ts

mentally updated and merged. Moreover, updating or merging tasks are evaluated by storing the SFs with 7 different resolutions. Figure 3.25 and 3.24 illustrate the results for merging and updating task. The time cost of the merging task is sensitive to the resolution but the updating task is not. In a high resolution, the merging task is more efficient than the updating task. The reason is that the updating task uses the time-consuming operator *encode* but the merging task does not.

An Illustrative Case Study

To demonstrate how *META* facilitates summarization, we list 3 tasks (1st row) as well as the corresponding statement (2nd row) in Figure 3.26 to show how the analysts work on *security-win* dataset. We also attach corresponding summarization results (3rd row) by implementing the *describe* operator according to [JPL11]².

As shown in Figure 3.26, the analysts only need to write one or two commands for each task. All the details are handled by the framework. Besides convenience, *META* also improves the reusability of data due to the SF's natural property. Once

²source code is available at <http://users.cs.fiu.edu/~yjian004/#codes>

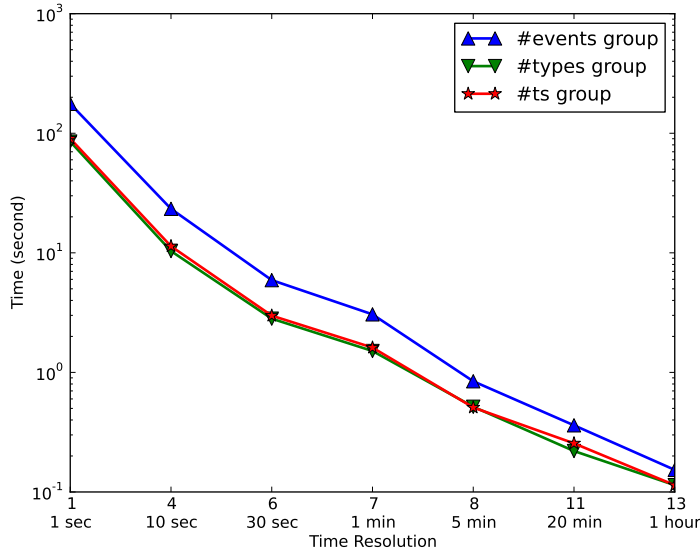


Figure 3.24: Updating task

the security-win log is stored in SF, it is directly available for all the 3 tasks, and there is no need to generate or maintain any intermediate data.

Without *META*, the analysts need to write programs on their own to conduct the data transformation and extraction. Taking task 2 for instance, the analysts should write several programs to transform the events in hourly resolution, to pick out the records related to the event types 538, 540, 576, 858, 861, and to extract the records occurring between 11/01/2011 and 1/29/2011. The analysts would do similar tedious work when facing the other two tasks.

3.4 Chapter Summary

In this chapter, we have discussed the solutions to facilitate event summarization, including a novel event summarization algorithm and a general summarization framework.

In terms of event summarization algorithm, a method called *Natural Event Summarization* has been proposed. *NES* summarizes the event sequences from the per-

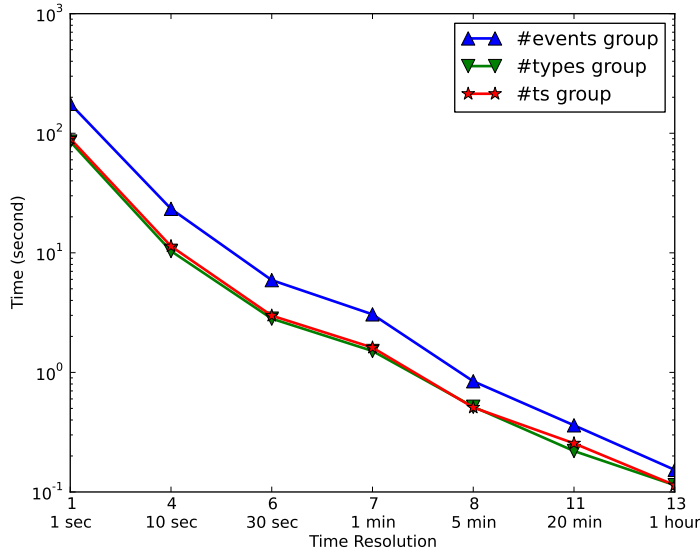


Figure 3.25: Merging task

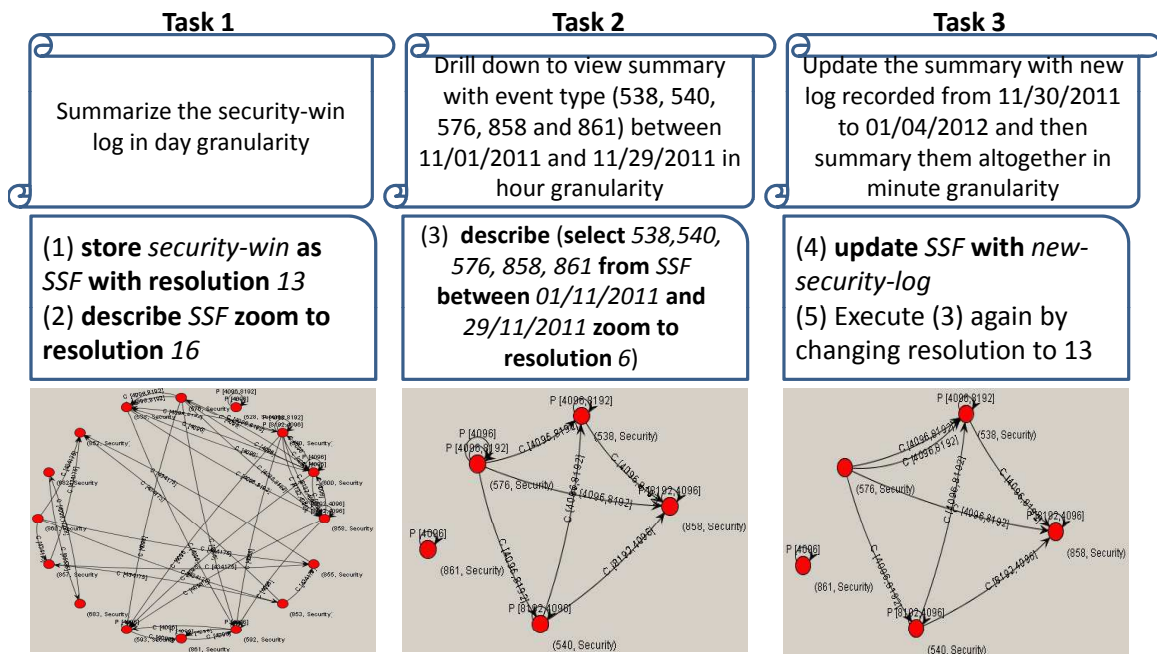


Figure 3.26: Summarize with META

spective of depicting the event patterns by capturing the temporal relationships among same-type and different-type of events. It is able to find the best set of disjoint histograms to summarize the input event sequence based on *MDL*. Moreover, to improve the efficiency, a heuristic boundary pruning algorithm has been proposed

to prune the unlikely boundaries and reduce the size of the histogram graph. Finally, *ERN* is used to present the final event summary in a more interpretable way.

In terms of event summarization framework, a general multi-resolution framework called *META* has been proposed. In *META*, the events are stored in the form of summarization forest. Moreover, a set of atomic operations on top of the data model and a set of summarization tasks has been proposed to ease the work of the analysts. To demonstrate the effectiveness of *META*, experimental evaluation has been conducted and the results are positive.

Temporal Mining for Cloud Demand Prediction

In this chapter, we will focus on the problem of leveraging temporal mining techniques to facilitate the operation and management of cloud services and systems. The application of temporal mining techniques to the domain of cloud services is able to increase the user satisfaction and help the service providers to better manage their systems. Concretely, we will focus on a specific problem – cloud system demand prediction, aiming to make the cloud systems to be more self-adaptive.

In this chapter, part of the content in this section has been published during my Ph.D study, including the problem formulation and the proposed solutions. The outline of this chapter is as follows: The motivation and challenges of this topic will be presented in Section 4.1; In Section 4.2, the cloud demand prediction problem will be formulated as a time series prediction problem; After that, the detailed solution of this problem will be introduced in Section 4.3 and 4.4 respectively; Finally, the conclusion will be given in Section 4.6.

4.1 Motivation and Challenges

Cloud service gradually becomes the ubiquitous choice for modern IT-solution in business activities. The paradigms such as *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a Service (SaaS)* are able to provide different styles of services to cater the taste of service customers with different requirements and needs. Among these paradigms, *IaaS* is the most fundamental service which is both elastic and economical. *IaaS* enables the cloud service customers to dynamically request proper amount of virtual resources (in terms of virtual machines) according to their actual business requirements.

Due to the pay-as-you-go promises made by the cloud service providers, the resource utilization becomes more flexible than the traditional fixed price charging

style. As the resource capacity planning burden has been shifted to the cloud service providers, then need to carefully prepare the available resources to both satisfy the service customers' needs and reduce the unnecessary resource waste. Under such circumstance, the techniques for the problems *effective cloud capacity management* and *instant on-demand VM provisioning* are crucial for these cloud services providers. In the next part, we will explain how difficult the above two problems are, and then we will introduce the corresponding solutions for them in details.

4.1.1 Challenges for Cloud Capacity Management

Resource capacity planning in traditional IT services is straightforward. This is because once a customer requires a service, the service terminate time is known to the service provider. In such scenario, the service vendors can simply upgrade the data centers by continuously scaling up the infrastructure to cater the increasing demands.

Capacity planning for cloud services is different and cannot be considered as a trivial task. Due to the demand fluctuates, the available resources in the cloud are difficult to be always fully utilized.

When the capacity of the cloud is overestimated by the service provider, the extra prepared but unused resources would be wasted. In nowadays, even a small portion of such overestimation would result in a large amount of waste. Due to the increasing scales of cloud infrastructure, the issue of energy consumption of cloud computing has gradually drawn more and more attention. As is reported, The US Environment Protection Agency (EPA) estimates that the energy usage at cloud data centers is successively doubling every five years. In the year of 2011, the annual electricity cost at these data centers would be approximately \$7.4 billion [PMSR09]. It is also reported that the direct monthly energy cost for data centers makes up more than 23% of the total amortized costs. If the indirect energy costs such as the power supply

infrastructure and cooling infrastructure are included, the energy consumption related to cloud computing would make up 42% of the total cost of the cloud systems [Ham09]. Moreover, unused physical resources not only cause the energy waste, but also result in more early purchase costs and environment pollution. As the price of the same equipment is always decreasing, later the equipment is purchased, lower its price would be. Furthermore, the overestimated of resources would cause extra associated cost such as network, labor, and maintenance, all of which are proportional to the scale of the infrastructure and therefore is non-trivial [GHM⁺09]. On the other hand, the underestimated of the cloud capacity would degrade the user satisfaction as it would cause resource shortage, which will further result in revenue loss. As non-trivial waiting time is needed for the on-demand request, the cloud has to postpone serving new customers if the actual demand is higher than the existing capacity. Once the shortage is severe, even existing customers would be affected, resulting in defeating the promise that application in cloud can scaling-up whenever the workload increases.

4.1.2 Challenges for On-demand Virtual Machine Provisioning

The volatility of cloud service makes the VM provisioning and de-provisioning occur frequently. It is true that state-of-the-art VM provisioning technology is able to provision a VM within a couple of minutes. Such time delay is affordable for normal services but would be fatal to the time-sensitive services which require real time scaling. However, from the infrastructure perspective, there is little hope that potential new cloud-specific devices can immediately and significantly reduce the provisioning time. It is true that the streamlining VM technology [LMT⁺04] allows the cloud service users to preview and use the VM before it is entirely prepared, but such tech-

nique still not be able to allow the users to use the demanded VM before an enough portion of it is ready.

From the perspective of business operation, a simple yet effective solution to reduce the provisioning time is to ask all the users to provide their schedule of using VM ahead of time. However, the idea is not likely to be used for many practical reasons:

- The promise of cloud service is to provide the virtual resources whenever the users need it. Such requirement would contradict the motivation of cloud computing because they constrain the behaviors of the users. Also, it is not likely that the users would like to provide their schedule ahead of time.
- The service customers themselves might not be able to set a schedule describing how many resources they exactly need in the future. It is because the demand is associated with the actual business activity in the future.
- As contract is not necessary for cloud services, the constituents of customers are always changing. New customers can join in and old customers can leave at any time.
- Even if the service customers are willing to provide their schedules. The actual schedules may change at any time and the existing schedule would be out of date.

Due to these business constraints and technology limitations, it is conceivable that resolving the problem of effective cloud resource provisioning is inherent difficult. Similar to the cloud capacity planning scenario, different types of VM mis-provisioning would cause different consequences. If a service customer sends a request for a certain type of VM and there is no prepared VM that matches this request, the cloud needs to provision the VM on-the-fly, which is time-consuming. In such situation, the *Service Level Agreement (SLA)* would be violated and the penalty would be very high. Usually, the penalty is in form of money and can be as high as several thousand

dollars. On the other hand, if the prepared VM is not consumed in the end, the associated resources are wasted. Moreover, the wasted VMs would indirectly occupy the resources that can be allocated to other useful VMs. If the mis-provisioning is severe, although the workload of the cloud is high, its effective utility is low.

For the remaining of this chapter, we would investigate this problem from the angle of time series prediction so that the demand can be predicted and the resources can be prepared in advance.

4.2 Problem Formulation

In principle, both the problems of *effective cloud capacity management* and *instant on-demand VM provisioning* are about preparing the virtual resources properly. They both aim to ensure that the prepared resources can match the real demand in the near future.

To uniformly formulate these two problems, we use the number of *VM unit* to quantify the amount of resources that is needed in a time slot (We will discuss how we choose the proper time slot in later sections.). The *VM unit* is defined as the basic and smallest unit of virtual resource, which is associated with a set of certain amount of physical resources such as CPU time, main memory, storage space, electricity etc. In real cloud systems, any virtual resource a customer can apply should be a multiple of the *VM unit*. For example, IBM's cloud product SCE defines one 64-bit *VM unit* as one 1.25 GHz virtual CPU with 2G main memory and 60G of storage space. The customer can request VMs with different multiples of the basic *VM unit*, such as *Copper*, *Bronze*, *Silver*, *Gold*, and *Platinum*. For the remaining of this thesis, the amount of needed resources is quantified by the number of *VM units* by default.

Let v_t (the number of *VM unit*) be the actually required virtual resource at a future time t , and \hat{v}_t be the corresponding prepared resources at time t , the goal of effective capacity planning and instant provisioning is to prepare the needed resources at each time slot t , such that:

$$E = \sum_t f(v_t, \hat{v}_t) \quad (4.1)$$

is minimized. In Equation (4.1), $f(\cdot, \cdot)$ represents an arbitrary cost function that is used to quantify the prediction error. A good resource predictor should be able to reduce the error as much as possible.

Specifically, in the context of cloud capacity planning, v semantically denotes the quantified amount of physical machines, associated available cooling systems, corresponding power supplies, and the number of system operators working for the cloud. In the context of instant VM provisioning, v represents the number of VM instances with a certain type that should be prepared before the customers actually send the requests. As we focus on the IaaS paradigm resource provisioning in this thesis, each VM is associated with a certain operating system or middle-ware, i.e. *Linux Red Hat Enterprise 5.5*, *Windows Server 2003*, etc.

4.3 Cloud Demand Prediction System Framework

According to the problem description mentioned above, a natural solution for the formulated problem is to leverage the state-of-the-art time series prediction techniques to predict the future resource needed. However, based on our empirical study about the available *Smart Cloud Enterprise* log data, the following difficulties are found to make the task non-trivial:

1. **The resource demands are highly unstable.** This is caused by two characteristics of cloud services: *the unstable customer constituents* and *the freestyle of resource acquisition/releasing*. In the presence of dynamic varying customer group, both the temporal characteristics of the cloud capacity and the required VMs for each type may contain random factors that can mislead the prediction algorithms. Figure 4.1 shows the change of the customer number over time¹. This figure implies that the number of customers is continually increasing. Therefore, even the old customer keep their request behavior, the overall request demands still change over time. This phenomena would cause the distributions of the resource demands to be unstable. Figure 4.2 illustrates the request history of three frequent requested customers. As is shown, these time series share no common pattern from each other.
2. **To enable instant provisioning, we need to predict the demand of each VM type separately, and then prepare them accordingly.** Since each type of VM has different characteristics in terms of demand amount, VM life-time, degrees of emergence, and physical resource requirements, their corresponding time series (See Figure 4.3) also show different temporal properties such as scale, degree of sharpness, length of cycle, degree of fluctuation, etc. As the data is divided into multiple chunks, it is more difficult to conduct the prediction with less data.

To properly address the above difficulties, we design and implement a real time prediction system that enables the automatic cloud management. At the core of the proposed system, we propose the ensemble time series predictor that combines the prediction power of a set of state-of-art prediction techniques. Besides combining individual predictors, we also consider the temporal correlations to ameliorate the

¹For confidential issue of the data provider, we remove y-axis in all the time series.

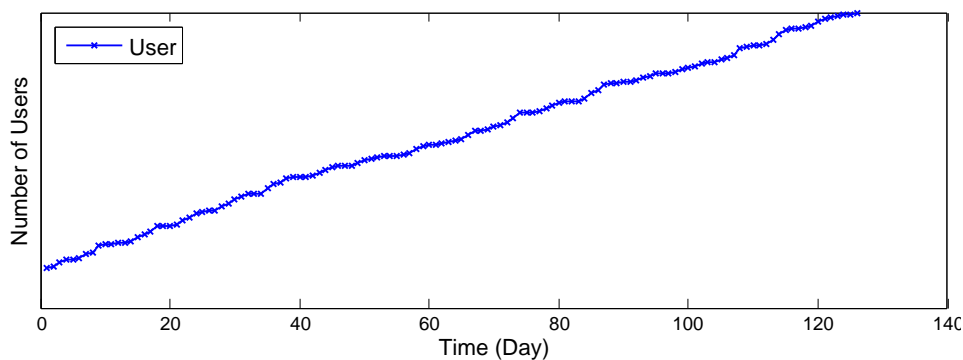


Figure 4.1: The dynamics of customers

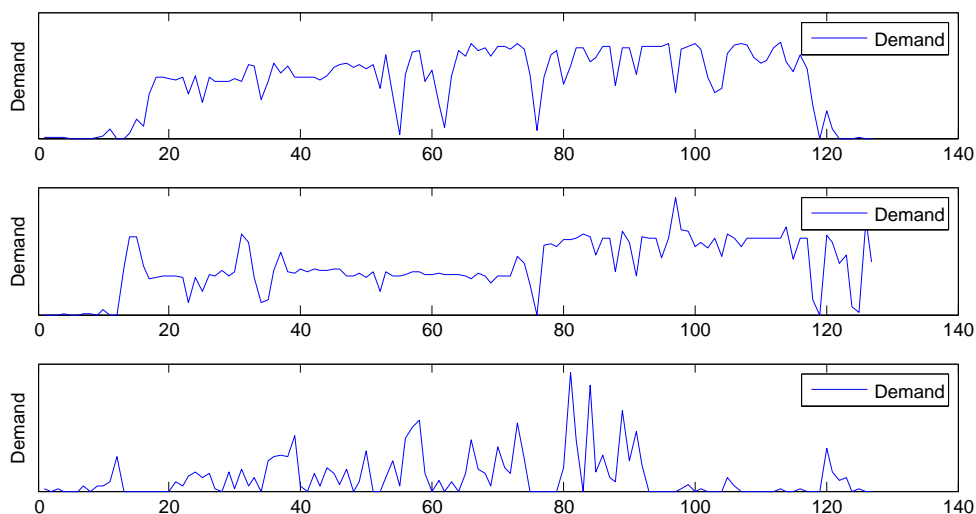


Figure 4.2: Time series of resource demands grouped by customer. The three time series show the demand of three frequent requesting customers. (The scales of all the time series are not normalized for ease of visualization.)

prediction. Figure 4.4 illustrate the framework of our system. As is depicted, our system works as an associated system for the cloud. It mainly contains following modules:

1. *Data Preprocessing Module.* This module conducts the data preprocessing works such as data cleaning, data transformation, and feature extraction. It is responsible for: (1) filtering out the unnecessary and unimportant information from the raw data; and (2) extracting the high-level characteristics of the filtered data and transform them into time series.

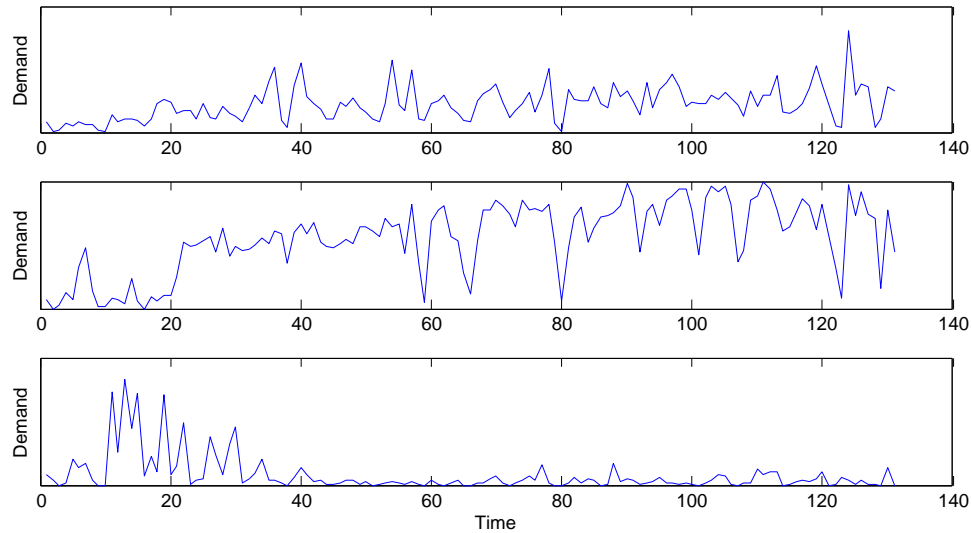


Figure 4.3: Time series of resource demands grouped by VM types. The three time series show the demand of three frequent requested VM types. (The scales of all the time series are not normalized for ease of visualization. For confidential issue, the value on y-axis are removed.)

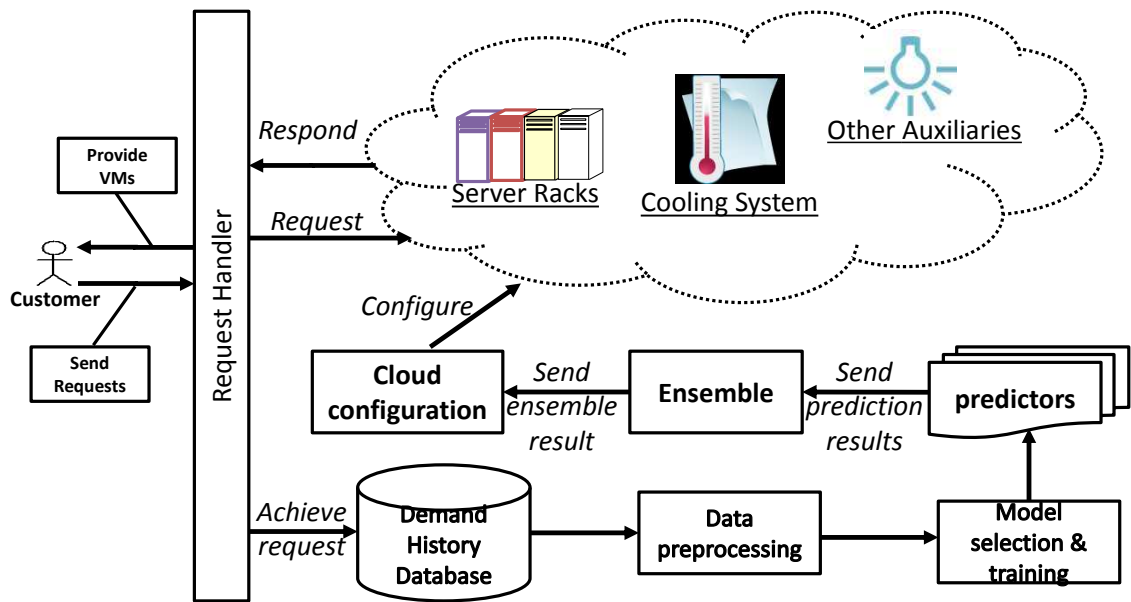


Figure 4.4: Cloud provisioning prediction system framework

2. *Model Generation Module.* This module is responsible for building the models of separate predictors. It periodically selects and trains the most suitable models of each predictor based on the latest requests. Once the work is finished, the

new parameters of the trained models are stored into a particular file and the demand prediction module is notified.

3. *Ensemble Prediction and Adjustment Module.* Once the individual prediction models are built, they are used to predict the future demands separately. Afterwards, an ensemble mechanism is used to obtain the final prediction results, and then the results are sent to the configuration module. Finally, if necessary, the temporal correlations are considered to help adjust the prediction result.
4. *Configuration Module.* This module is responsible to configure the cloud based on the prediction results. It knows all the running status of the cloud, including the amount of available computing resources, the workload of cooling system, the power supply etc and will dynamically adjust the cloud capacity based on the current status and the prediction results.

4.4 Cloud Demand Prediction Approach

In this section, we will first propose a novel measure to quantify the prediction result error. Then we introduce the approaches we used to predict the capacity and the VM provisioning demand, respectively.

4.4.1 Prediction Result Evaluation Criteria

Traditional regression cost measure such as *mean average error (MAE)*, *least square error (LSE)*, and *mean absolute percentage error (MAPE)* are all symmetric measures [Cha03]. These measures are not perfectly appropriate to quantify the prediction error in cloud demand prediction scenario. This is because the consequence of

over-prediction and under-prediction are semantically different, and they result in heterogeneous costs.

In cloud scenario, if the prepared resources are more than the actual demand, the service provider will suffer. This is because the idled and wasted resources are not billable. On the other hand, if the prepared resources are less than the actual demands, the service quality of the customers would be affected and the SLA would be violated.

Considering the characteristics of the cloud, we propose a novel cost measure called *Cloud Prediction Cost (CPC)* to quantify the quality of prediction results. *CPC* is an asymmetric and heterogeneous measure that models the under-prediction and over-prediction differently: the cost of SLA penalty and the cost of idled resources. Respectively, we use $P(v^{(t)}, \hat{v}^{(t)})$ and $R(v^{(t)}, \hat{v}^{(t)})$ to quantify their costs. The total costs can be represented as:

$$C = \beta P(v^{(t)}, \hat{v}^{(t)}) + (1 - \beta)R(v^{(t)}, \hat{v}^{(t)}), \quad (4.2)$$

where β is the weight to tune the importance between these two types of costs. Take the capacity prediction scenario for instance, if the workload (the proportion of current capacity to the maximum capacity) of the cloud is low, the system administrator can increase β , and thus the cost measure focusing more on the SLA penalty.

CPC is a generic cost measure that can be used for both capacity management and instant VM provisioning scenarios. Since different cloud has different objectives, the concrete quantification of the costs may vary accordingly. Basically, the R function and P function can be defined in any form that satisfy the following two properties:

1. *Non-negativity.* Both $P(v^{(t)}, \hat{v}^{(t)}) \geq 0$ and $R(v^{(t)}, \hat{v}^{(t)}) \geq 0$ should be hold for arbitrary non-negative $v^{(t)}$ and $\hat{v}^{(t)}$.

2. *Consistency.* If $v_1(t) - \hat{v}_1(t) \geq v_2(t) - \hat{v}_2(t)$, then $P(v_1(t), \hat{v}_1(t)) - P(v_2(t), \hat{v}_2(t))$ is either consistently positive or negative. Similarly, if $v_1(t) - \hat{v}_1(t) \leq v_2(t) - \hat{v}_2(t)$, then $R(v_1(t), \hat{v}_1(t)) - R(v_2(t), \hat{v}_2(t))$ is either consistently positive or negative.

Cost of SLA penalty

The cost of SLA penalty is used to quantify the satisfaction of the customers. This cost is largely due to the under-estimation of the future resource requests. For simplicity, we use *request fulfillment time* (T_{avail}) to quantify the SLA penalty. On one hand, when available resources are enough, the new requests can be immediately fulfilled to the requester. On the other hand, if there is not enough resources, the requester has to wait for T_{delay} until new resources are available and allocated. The waiting time is undecidable, but typically $T_{avail} \ll T_{delay}$ since this situation always involves procedures like garbage collection and resource reallocation.

In our system, we quantify the SLA penalty with P function and its definition can be seen as Equation 4.3. Based on the definition, the penalty is proportional to the severity of under-estimation. More sophisticated forms of the P function can also be used. For example, a common SLA typically specifies a penalty threshold for resource fulfillment time. The cost of a non-violated request in this case would have zero value for P function.

$$\begin{aligned}
 P(v^{(t)}, \hat{v}^{(t)}) &= \min(v^{(t)}, \hat{v}^{(t)})T_{avail} \\
 &+ \max(0, v^{(t)} - \hat{v}^{(t)})T_{delay}.
 \end{aligned}
 \tag{4.3}$$

Cost of resource waste

This is the non-billable part on the service vendor side. It includes the server aging, electricity waste and labor cost, etc. R_{vm} is used to denote the cost of waste caused by one *VM unit*. The R function is defined as Equation 4.4.

$$R(v^{(t)}, \hat{v}^{(t)}) = \max(0, \hat{v}^{(t)} - v^{(t)})R_{vm}. \quad (4.4)$$

Combining Equation 4.3 and Equation 4.4, the total resource prediction error is quantified as Equation 4.5.

$$\begin{aligned} C = f(v^{(t)}, \hat{v}^{(t)}) &= \beta P(v^{(t)}, \hat{v}^{(t)}) + (1 - \beta)R(v^{(t)}, \hat{v}^{(t)}) \\ &= \begin{cases} \beta v^{(t)}T_{avail} + (1 - \beta)(\hat{v}^{(t)} - v^{(t)})R_{vm}, & \text{if } \hat{v}^{(t)} \geq v^{(t)} \\ \beta(\hat{v}^{(t)}T_{avail} + (v^{(t)} - \hat{v}^{(t)})T_{delay}), & \text{if } \hat{v}^{(t)} < v^{(t)} \end{cases} \end{aligned} \quad (4.5)$$

For different clouds, the trade-off between SLA penalty and idled resources can be different. We use the parameter β to tune the importance between P function and R function. As mentioned in Section 4.2, we aim to minimize the total cost quantified by Equation 4.1.

4.4.2 Ensemble Prediction Model

To incorporate the prediction power of individual prediction algorithm, we utilize ensemble prediction techniques to handle most of the prediction tasks. The ensemble prediction is motivated by its classification counterpart but it has different property. In prediction scenario, temporal correlation is contained between records, while in classification scenario the data is assumed to be i.i.d (independent and identically distributed) [HTF09]. Moreover, the labels of prediction are continuous, so the scale of the ensemble results should be well controlled.

In ensemble prediction, we employ five different time series prediction techniques as shown in Table 4.1. To combine their prediction result, we propose a weighted linear combination strategy. Suppose the predicted value for predictor $p \in \mathcal{P}$ at time t is $\hat{v}_p^{(t)}$ and its corresponding weight at time t is $w_p^{(t)}$, the predicted value for a certain VM type at time t is

Method Name	Description
Moving Average	Naive Predictor
Auto Regression	Linear Regression
Artificial Neural Network	Non-linear Regression
Support Vector Machine	Linear Learner with Non-linear Kernel
Gene Expression Programming	Heuristic Algorithm

Table 4.1: Time series prediction techniques used for ensemble

$$\hat{v}^{(t)} = \sum_p w_p^{(t)} \hat{v}_p^{(t)}, \text{ subject to } \sum_p w_p^{(t)} = 1. \quad (4.6)$$

Initially ($t = 0$), all the individual predictors have the same contribution to the prediction result, i.e. $w_p^{(0)} = \frac{1}{|P|}$. The weight updating strategy for the prediction based ensemble is also different from the traditional classification based strategy. In classification scenario, the results can only be “correct” or “incorrect”, and the ensemble just needs to increase the weights of those correctly classified classifiers for weight updating. In the prediction scenario, the results are continuous values and the weights of the predictors would directly affect the ensemble result. Therefore the updating strategy should be carefully quantified.

We make use of the difference between each predicted value $\hat{v}_p^{(t)}$ and the real value $v^{(t)}$. In order to update the weights, we calculate the *relative error* $e_i^{(t)}$ caused by predictor i at time t according to

$$e_i^{(t)} = \frac{c_i^{(t)}}{\sum_p c_p^{(t)}} w_i^{(t)}, \quad (4.7)$$

where $c_i^{(t)}$ (or $c_p^{(t)}$) is the prediction cost of predictor i (or p) computed by the cost functions such as *MAE*, *LSE*, *MAPE* and *CPC*.

Note that the relative errors cannot be used as the new weights of the predictors since they are not normalized. As the final predicted value is the linear combination of all the results of individual predictors, Equation 4.8 should be used to normalize the weight.

$$w_i^{(t+1)} = \frac{e_i^{(t)}}{\sum_p e_p^{(t)}}. \quad (4.8)$$

It is easy to prove that the weight of the best predictor at each time is guaranteed to be increased by this weight update strategy.

4.4.3 Cloud Capacity Prediction

Time Series Selection

Before adopting ensemble prediction for the problem of capacity prediction, we need to pick a suitable type of time series first. Given the cloud trace data, three types of time series are available for capacity prediction: The original capacity time series (See Figure 4.5), the capacity change time series (See Figure 4.6), and the separated provisioning/de-provisioning time series (See Figure 4.7).

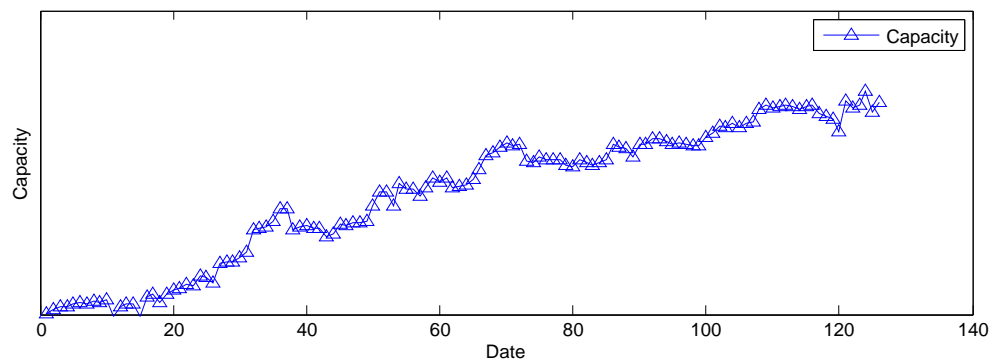


Figure 4.5: Original capacity time series

It is easy to observe that the capacity of the cloud gradually increases from the long-term perspective. The non-stationarity property of this time series makes most of the time series prediction models to be invalid. It is true that we can leverage *ARCH* [Eng82] to directly model and predict such non-stationary time series. However, *ARCH* is a parametric model that only performs well under stable conditions. The highly fluctuate request of the cloud service makes the prerequisite of this model

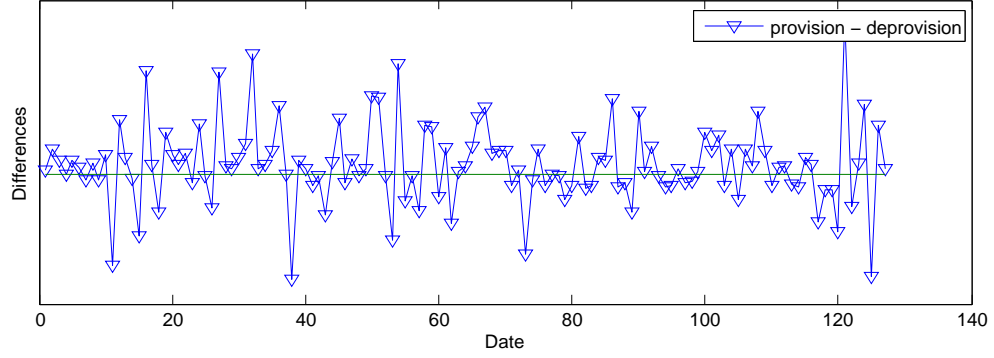


Figure 4.6: Capacity change time series

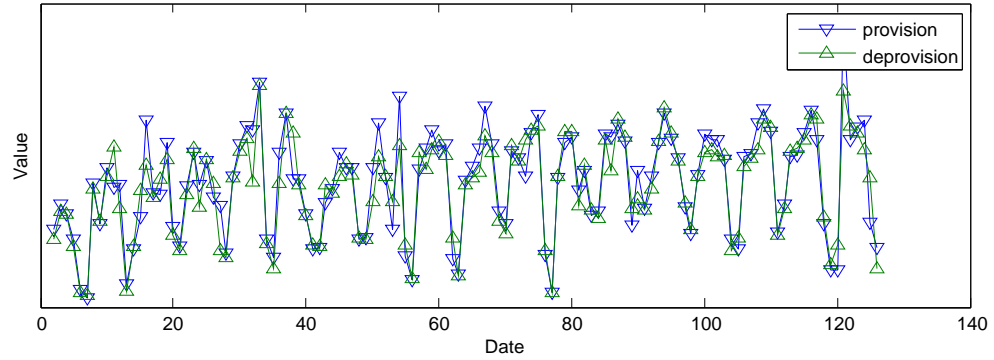


Figure 4.7: Provisioning/de-provisioning time series

unsatisfied. Furthermore, *ARCH* is based on symmetric cost functions. This makes it inappropriate to the cloud scenario.

The capacity change time series is obtained by taking the first derivative of the original capacity time series. This time series shows a stationary trend, and it is proper to be modeled by most of the time series prediction models. However, the irregularity of the trends implies that its temporal pattern is not easy to be discovered.

If we further decompose the capacity change time series into a provisioning component and a de-provisioning component, a weekly periodic pattern appears. This pattern implies that this type of time series is easier than the previous two. In our system, we utilize this kind of time series for prediction. Generally, *the capacity change can be estimated according to the difference between the provisioned and de-provisioned resources*, i.e. $\Delta = prov_t - deprov_t$, where Δ denotes the change of

capacity at each time slot, $prov_t$ and $deprov_t$ denote the quantified provisioned and de-provisioned resources.

Interpretation of cost functions

Over- and under-estimation have different consequences when estimating the provisioning and de-provisioning of resources. Table 4.2 illustrates the cost matrix for these two situations. For provisioning prediction, an over-estimation has no negative effect on the customer and only causes idled resources, but an under-estimation degrades the service quality. For de-provisioning prediction, the costs are reversed.

	Provisioning	De-provisioning
Over-estimation	resource waste	SLA penalty
Under-estimation	SLA penalty	resource waste

Table 4.2: Cost matrix for capacity estimation

Cost of SLA penalty (P function): In the scenario of capacity prediction, when there is not enough resource, the requester has to wait until new resources are available. New resources are supplemented by starting new servers at the back-end of the cloud. Therefore, T_{delay} is used to indicate the waiting time plus the physical server cold start time plus the VM fulfillment time, i.e. $T_{delay} = T_{wait} + T_{start} + T_{vm}$. On the other hand, if available resources are enough for current request, the resources associated to the request can be immediately allocated and the provisioning procedure can immediately start. In this situation T_{avail} only include the VM fulfillment time T_{vm} . In our system, we define the P function for provisioning prediction syntactically the same as Equation 4.3. The P function for de-provisioning prediction can be obtained by switching the variables $v^{(t)}$ and $\hat{v}^{(t)}$.

Cost of resource waste (R function): Syntactically, the R function in the generic cost function is the same as the one for capacity prediction. Semantically, R_{vm} in R function is used to denote the waste of physical server plus the amortized

waste of cooling system, electricity supply, and human labor, etc. Also, the R function for de-provisioning prediction can be obtained by exchanging $v^{(t)}$ and $\hat{v}^{(t)}$.

A Better Prediction Approach for De-provisioning

The future de-provisioned VM are bounded by the number of total used VMs in the cloud. At any time, any customer cannot de-provision more VMs than they requested. The ensemble technique is workable for de-provisioning prediction, but we have an alternative way to estimate the de-provisioning in a more intuitive way [JPLC12a]. We will first introduce this method and then show that the new proposed method can achieve better accuracy than the ensemble prediction approach through experiments in Section 4.5.

Since we have all the information of the potential de-provisioned VMs, we can build profiles using the temporal dynamics for the VM image types. Rather than the observed time series, the temporal characteristics provides more interpretable context of the de-provisioning that can facilitate the estimation. Our empirical exploration demonstrate that knowing the current life time of individual VM is helpful for estimating the de-provisioning [JPLC12b]. That is, we can infer when a certain VM would be de-provisioned through the life time distribution of the images. Since the true distribution of the image life time is not available, we need to estimate it from the historical data first.

To investigate whether the VM life time depends on the time when it is requested, we divide the requests into two groups by putting the requests recorded during Monday and Wednesday to the first group and the remains to the second group. Then we conduct statistical test on these two datasets. The result indicates that we can safely assume that the VM life time distribution does not depend on the time they are requested. Suppose $life(VM)$ is the current life time of a VM, and n_i is the frequency

of VMs with life time t_i . We estimate the empirical cumulative distribution function (CDF) of the life time with a step-wise function in Equation 4.9.

$$\hat{F}(x) = P(\text{life}(VM) \leq x)$$

$$= \begin{cases} n_1 / \sum_i n_i, & t_1 \leq t < t_2, \\ (n_1 + n_2) / \sum_i n_{t_i}, & t_2 \leq t < t_3, \\ \dots, & \dots \\ (\sum_{i=1}^{n-1} n_i) / \sum_i n_i, & t_n \leq t. \end{cases} \quad (4.9)$$

The output of the estimated CDF denotes the probability of a VM that would be de-provisioned when its current life time is t_i . Utilizing $\hat{F}(x)$, the de-provisioning demand can be estimated by

$$\sum_{i \in VM_{active}} \hat{F}(\text{life}(i) \leq t_{now} - t_{start}(i)), \quad (4.10)$$

where VM_{active} denotes the set of VMs currently used, t_{now} denotes the current time and $t_{start}(i)$ denotes the provisioned time of VM i .

4.4.4 Virtual Machine Type-aware Provisioning Prediction

In this section, we firstly introduce how we adopt the generic cost function according to the instant VM provisioning scenario. Then we introduce how we reuse the prediction techniques with minor modification to solve the instant VM provisioning problem.

Interpretation of cost function

The cost function for the instant VM provisioning scenario has the same form with the generic cost function, but they have different semantics. The P function and R

function represent the *the cost of provisioning delay* and *the cost of idled VM* in the instant provisioning scenario.

When a request for a particular type of VM arrives, if there is no VM with the same type prepared in the cloud pool, the cloud has to provision the VM on-the-fly. We denote the on-the-fly provisioning delay as $T_{delay} = T_{miss}$, which is similar to the “miss” in system cache scenario. On the other hand, if the requested VM has already prepared in the pool, the cloud can simply transfer its ownership to the customer and immediately finish the provisioning procedure. In this case, the request is “hit” and the delay is $T_{avail} = T_{hit}$. Typically, $T_{hit} \ll T_{miss}$. A “hit” request can be handled in seconds, but the on-the-fly provisioning time for a “miss” request could be up to minutes for the state-of-art cloud systems. If the requested VM is configured with complex service components, a missed request would take even longer due to the software integrity, security checking, and sometimes other manual processes.

When the provisioning of a certain VM type is over-estimated, part of the prepared VMs would be wasted. The service vendor would be responsible for the cost of this part of VMs, since they cannot be charged to the customers. Same as in capacity prediction scenario, the cost of an idled VM can be quantified by the number of *VM units* and we can use R_{vm} to denote its cost. Therefore, the over-predicted cost can be quantified by the same equation (Equation 4.4) used in capacity prediction scenario.

Prediction of individual VM provisioning

Different from the provisioning and de-provisioning prediction for capacity prediction, to enable instant VM provisioning, we need to separately handle the VM provisioning time series of each VM type. As shown in Figure 4.8, the time series of individual type requests demonstrates high irregularity comparing with the total requests time series. This phenomenon implies that the auto-correlation of these time series is more difficult to discover. Therefore, besides the auto-correlation, we also exploit

the correlation between different VM types to mitigate the risk of serious prediction deviation. For instance, the requests time series of the same series of VM type, i.e. *Windows Server 2003* and *Windows Server 2005*, are highly correlated. We called this procedure *Correlation Ensemble*.

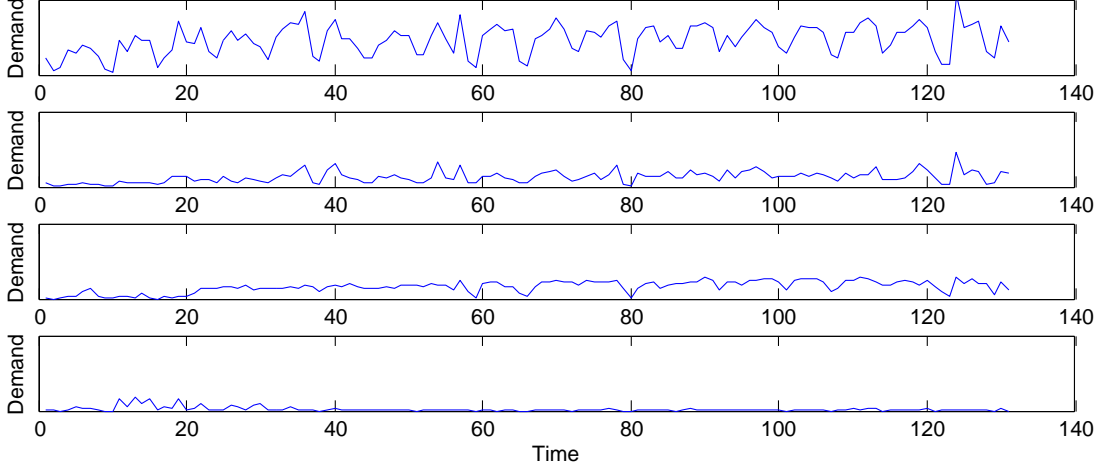


Figure 4.8: Total requests time series and individual VM type requests time series

In our system, we utilize the correlation matrix of time series to help post-process the prediction results. Suppose $\sum_{(t)}$ is the covariance matrix of VM types at time t , $cov_{ij}^{(t)}$ denotes the covariance between resource types i and its j th correlated resource. By considering the positive influence of the strongly correlated time series (in our system, we set correlation as 0.8, based on the empirical study), the prediction value $\hat{u}_i^{(t)}$ of time series i at time t now becomes

$$\hat{u}_i^{(t)} = \frac{\sum_{j=1}^k cov_{ij}^{(t-1)} s_{ij} \hat{v}_k^{(t)}}{\sum_{j=1}^k s_{ij} cov_{ij}}, \quad (4.11)$$

where $s_{ij} = \bar{t}_i / \bar{t}_j$ denotes the scale difference between two time series and k is the number of strongly correlated time series. In our current design, we only considered the positively correlated time series, the negative influence consideration is one of our future works.

To further mitigate the cost of mis-prediction caused by the inherent prediction difficulty, we introduce a module called *Reservation Controller*. Its function is to reserve the unused VMs and only notifies the cloud to prepare new VMs when all the reserves of the VM type are used up. Reservation Controller provides a good buffer mechanism that effectively reduces the waste of VMs. It can be integrated into the cloud configuration module.

4.5 Experimental Evaluation

To evaluate the effectiveness of our system, we use the real VM trace log of IBM Smart Cloud Enterprise (SCE) to conduct the experiments. The trace data we obtained records the VM requests for more than 4 months (from late March 2011 to July 2011), and it contains tens of thousands of request records with more than 100 different VM types. In this trace data, each request record contains 21 features such as *Customer ID*, *VM Type*, *Request Start Time*, and *Request End Time*, etc. The goal of the experimental evaluation is to answer the following questions:

- Whether our prediction mechanism for capacity planning and instant VM provisioning reliable?
- Whether the measure *CPC* is practical and flexible?
- To what extent can our system decrease the average request fulfillment time for both problems?

4.5.1 Experiment Setup and Pre-processing

Data preprocessing is the step before data modeling and prediction. There are two reasons for data preprocessing of the raw trace data recorded by SCE.

1. The raw data contains useless request fields that would not be used during prediction.
2. The records of the requests are stored in a low-level representations. The requests need to be aggregated into proper granularity first, and then feed to the algorithm for prediction.

Feature Selection

Not all the 21 features of a request record are useful. In our current implementation, we only consider the *VM Type*, which illustrates the type of VM the customer requests; *Request Start Time*, which indicates the time that the customer sends the request; *Request End Time*, which indicates when the VM is released. Other features like *Customer ID*, *Customer Priority* and *Data Center* will be considered in our future work for personalized service quality improvement.

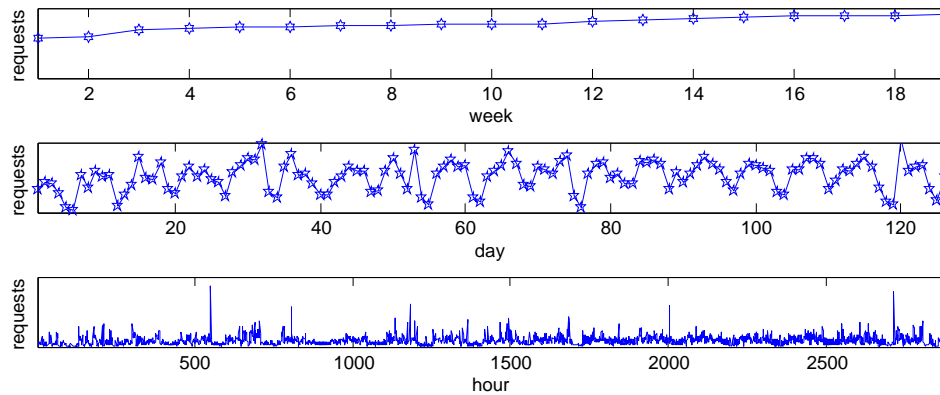


Figure 4.9: Time series with different granularities. From top to bottom, the time series are aggregated by week, day, and hour, respectively.

Time Series Aggregation Granularity Selection

All the time series we present in this paper are obtained through aggregating the raw trace records with a certain granularity. The time series aggregated by different granularities would have different levels of difficulty for prediction. For example,

Figure 4.9 shows the capacity provisioning time series aggregated by week, day, and hour, respectively.

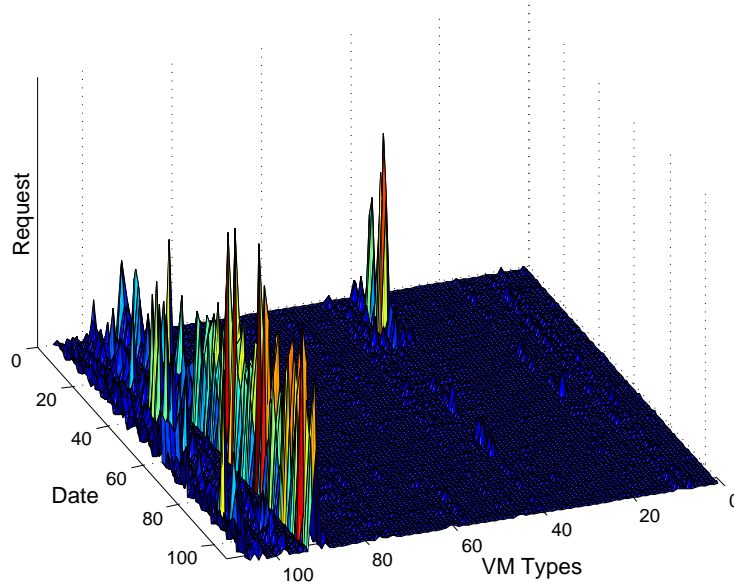


Figure 4.10: Request distribution in time-type-request view

This figure shows that the coarser the granularity, the larger the provisioning amount in each time slot. Therefore, the weekly aggregated time series requires the cloud to prepare the most VMs for each time slot. Compared with a finer granularity, a smaller portion of prediction deviation for weekly aggregated time series would result in a larger waste. Moreover, through exploration, we found the life time of most of the VMs is shorter than one week. Therefore, the weekly aggregated time series cannot reflect the real situation.

On the contrary, it is also not suitable to aggregate the records by hour, since the lifetime of the VMs is not so short. A too fine granularity would make the value on each timestamp lacks statistical significance. Table 4.3 list the results by measuring the irregularity of these time series in different perspectives with *Coefficient of Variance (CV)*, *Skewness*, and *Kurtosis* [HTF09]. Higher *CV* indicates larger volatility, higher *Skewness* indicates stronger asymmetry, and higher *Kurtosis* means more of the variance is the result of infrequent extreme deviations. In our comparison, the

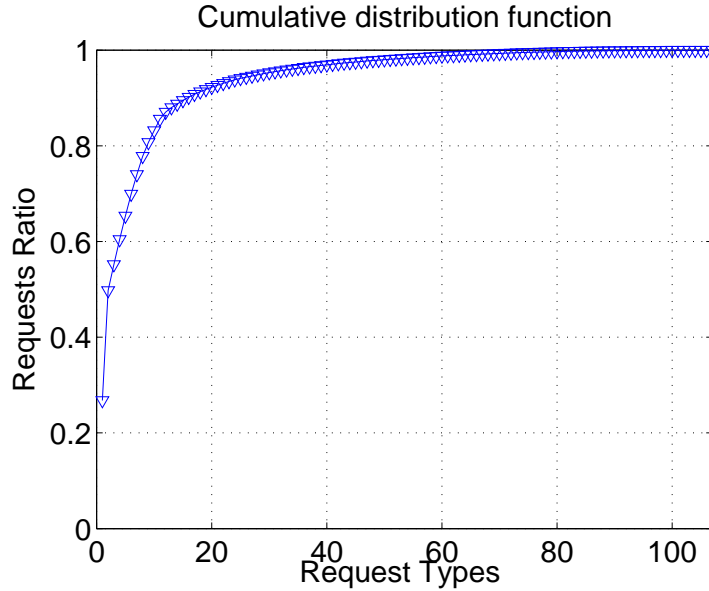


Figure 4.11: CDF of VM types

time series aggregated by hour has the largest values in all the three measures, indicating the hour granularity is not suitable to aggregate the time series. Therefore, based on above investigation, we aggregate the daily time series in our implementation.

Measure	Granularity	Week	Day	Hour
Coefficient of Variance		0.1241	0.4048	0.7182
Skewness		-0.5602	-0.2536	2.6765
Kurtosis		2.7595	2.5620	20.5293

Table 4.3: Statistics for time series irregularity

VM Type Selection

VM type selection is only relevant to instant VM provisioning problem. As mentioned in Section 4.4.4, we need to predict the future demand for each VM type separately. Figure 4.10 plots the distribution of requests in a time-type-request view. This figure clearly indicates one obvious characteristics: The distribution of VM request is highly uneven, and a small number of the VM types dominate the distribution. We also plot the corresponding cumulative distribution function (CDF) and rank the VM based

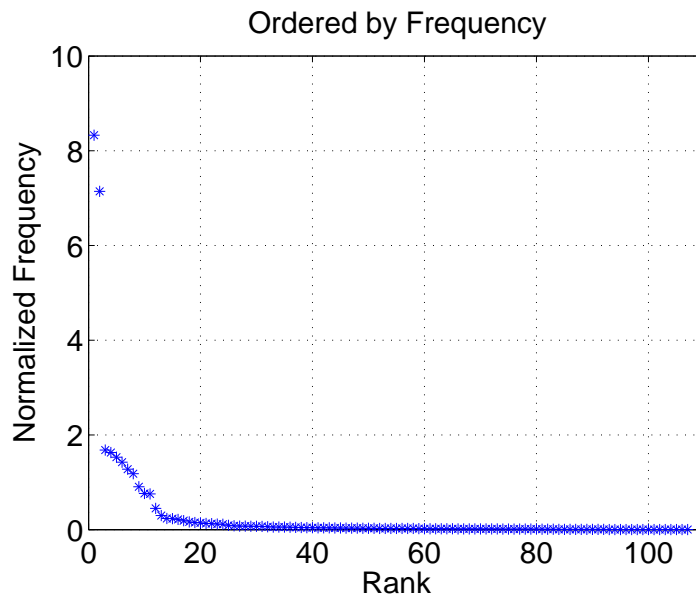


Figure 4.12: Frequency order of VM types

on their requests frequency in Figure 4.11 and Figure 4.12, respectively. The CDF shows that the VM requests obey the 80/20 rules — *more than 80% of the requests concentrates on less than 20% of the VM types*. In the frequency ranking plot, we observe that there is an inflection point between the 12th and 13th types, which explicitly divides the types into frequent and infrequent groups. The measures like *CV*, *Skewness* and *Kurtosis* on time series of these infrequent types also show higher values than those of frequent types, which demonstrate that the time series of these infrequent types are not regular enough to be modeled and predicted.

Notwithstanding the inherent prediction difficulty caused by the irregularity, the future demand of the infrequent VM types can be handled in a more empirical way. We design the data preprocessing module to periodically checks the change of the rank, and build prediction modules for the frequent VM types only. For the infrequent types, the cloud prepare a fixed number of VMs for each type. The fixed number is set as the moving average of the recent requests of the type. Once the prepared VMs are used up, the cloud just needs to prepare another batch of VMs.

4.5.2 Provisioning Prediction Evaluation

To evaluate the performance of provisioning prediction, we compare the accuracy of our ensemble predictor with the individual predictors mentioned in Table 4.1. In this experiment, we use *CPC* as the measure. For parameter setting, we set $\beta = 0.5$, $T_{delay} = 1200$ (the time for on-demand preparation, including server boot up, VM creation, security checking, patching, and configuration etc.), $T_{avail} = 10$ (resource is available almost instantly), and $R_{vm} = 500$ (the cost of wasted resources for one VM unit).

We partition the time series horizontally into two parts: the records before May 2011, and the records after May 2011. Then we use the data before May 2011 for training. The precision of these predictors are evaluated on the date of May, June, and July, respectively.

For each individual predictor, the parameter setting is as follows. *Random Guess* simply guesses the future demand by randomly picking a number between 0 and the maximum demand known so far. We set the sliding-window of *MA* and the number of variables in *AR* as 7, since we have observed a suspected weekly periodical pattern. For *Neural Network*, we leverage a 3-layer topology with 1 hidden layer. The neurons in input layer take the information of the latest 7 days as input variables, and the number of hidden layer and output layer is set as 7 and 1, respectively. This topology enables the neural network to capture any combinations of the input variables. As for *SVM*, we first leverage the grid search to identify the best parameter combinations with training dataset, and then we use regression *SVM* for prediction. For *GEP*, we set the population as 40, number of evolution generation as 1000, operator set as $\{+, -, \times, /, \sqrt{\cdot}, \sin\}$. To eliminate the randomness of some predictor (*Random Guess*, *Artificial Neural Network* and *Gene Expression Programming*), the results are computed by averaging 10 runs of each predictor. Table 4.4 lists the evaluation results

of all the predictors. The results clearly show that the best predictor is different for different test datasets. And on average, the ensemble method achieves the best performance.

Predictor	May	June	July	Average
Random Guess	2281555	3507600	3080320	2956491
Moving Average	1295550	1293620	1293620	1294263
Auto Regression	504912	760110	1047275	770765
Artificial Neural Network	980780	1095102	1577127	1217669
Gene Expression Programming	866117	640405	1037705	848075
Support Vector Machine	3746005	2199010	1147240	2364085
Ensemble	538302	626585	1072840	745909

Table 4.4: The cost of different predictors measured by *CPC*

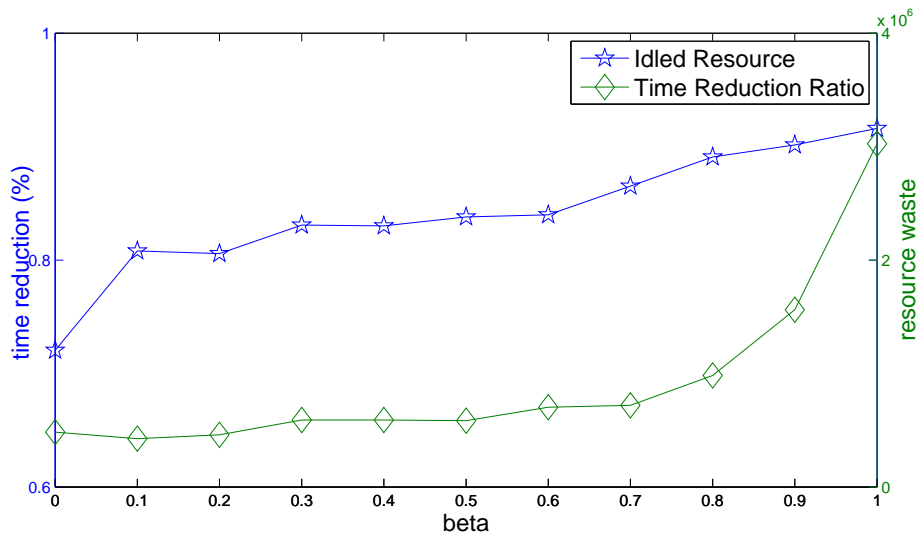


Figure 4.13: The effect of tuning β for capacity prediction

Effectiveness of Parameter Tuning

As mentioned before, the parameter β can be used to tune the preference of the predictor (optimistic vs. pessimistic). A higher β results in less SLA penalty risk (more time reduction) but increases the chance of resources waste. A lower β can reduce the idled resources risk but increases the chance of SLA violation.

Figure 4.13 illustrates how β affects the prediction results. In this figure, the x-axis denotes the value of β while y-axis denotes the average ratio of VM fulfillment time reduction (left) and the average cost of idled resource (right) each day quantified by R function.

As is shown, along with the increasing of β , both the time reduction ratio and the cost of idled resource cost increase. This is because when β is small (close to 0), the R function would have higher impact to the cost function than P function. In this case, the cloud would prepare relatively less resources to avoid the waste and does not take too much action to reduce the waiting time. On the contrary, when β is big (close to 1), the cost function will pay more attention to reduce the VM fulfillment time. In this case, the cloud would prepare relatively more resources to ensure the resource supply on the cost of higher chance of resource wasting.

4.5.3 De-provisioning Prediction Evaluation

We try four different methods to predict the de-provisioning demands based on the information of VM life time. For all these methods, the evaluation is conducted on the last 60 days of de-provisioning records, and all the other data are used as the training data. The details of these methods are listed as follows:

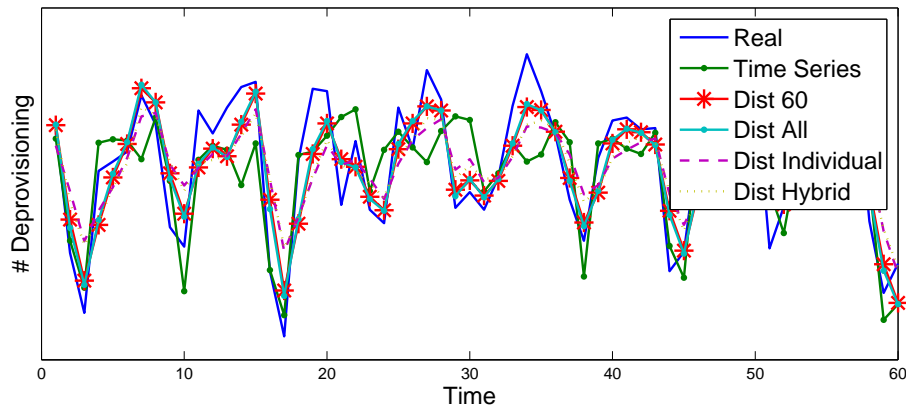


Figure 4.14: Prediction result of de-provisioning time series

Predictor	1st VM Type				2nd VM Type				3rd VM Type				Avg. Top 12
	MAE	MSE	MAPE	CPC	MAE	MSE	MAPE	CPC	MAE	MSE	MAPE	CPC	CPC
MA	40.3	2740.23	0.87	437305	51.53	4904.4	1.63	532132.5	7.23	88.5	1.67	76370	152402.29
AR	45.1	4034.57	1.52	300687.5	50.5	3893.63	5.32	391630	5.9	67.5	1.05	83292.5	137046.67
ANN	30.27	1203.07	0.67	388535	20.8	706.4	2.04	247085	4.97	29.57	2.14	37052.5	106113.96
GEP	17.37	1757.5	0.21	154440	54.77	5621.97	6.59	473347.5	4.1	27.63	1.59	33745	112149.79
SVM	68.03	5604.77	0.9	1010422.5	59.5	5578.37	2.89	890447.5	6.2	85.4	0.9	88737.5	234225.83
Ensemble	16.7	1212.1	0.21	158862.5	21.67	1057	2.04	254190	5.03	46.97	1.56	53327.5	88679.79

Table 4.5: The cost of prediction algorithms under different measurements. For the acronym, *MA* denotes *Moving Average*, *AR* denotes *Autoregression*, *ANN* denotes *Artificial Neural Network*, *GEP* denotes *Gene Expression Programming*, and *SVM* denotes *Support Vector Machine*

1. *Dist All*: This method leverages all the training data to estimate the global VM life time distribution by ignoring the VM type. For each day, it estimates the expected number of VMs that would be de-provisioned based on the probability of de-provisioning.
2. *Dist 60*: This method leverages the latest 60 days of training data to estimate the global VM life time distribution by ignoring the VM type. For prediction, it is the same as the first method.
3. *Dist Individual*: This method first estimates the life time distributions of each VM type, and then it predicts the expected number of de-provisioning VMs based on the life time distributions of corresponding types. This method is a finer granularity version of the first method.
4. *Dist Hybrid*: This method leverages all the training data to estimate both the global life time distribution and individual life time distribution. The expected number of de-provisioned VMs is calculated as $\alpha \hat{F}_i + (1 - \alpha) \hat{F}_g$, where α equals to the fraction between the frequency of specified image type and the most popular image type.

Besides these four methods, we also predict the future de-provisioning with the ensemble method and their prediction results (6 methods in total) are shown in Figure 4.14, where the x-axis denotes the time and y-axis denotes the number of de-provisioned VMs. As is illustrated, all of these methods successfully discover the

periodic patterns of the de-provisioning trend. However, the ensemble method has limited ability to fit the scale of the peaks. Among all these methods, the time series predicted by *Dist All* fits the real time series the best. Moreover, all these four life time distribution based methods have approximately the same accuracy and they all outperform the ensemble prediction method. The reason is that the methods *Dist All*, *Dist 60*, *Dist Individual*, and *Dist Hybrid* all estimate the de-provisioning from the empirical life time distributions, rather than simple inference from the observed time series.

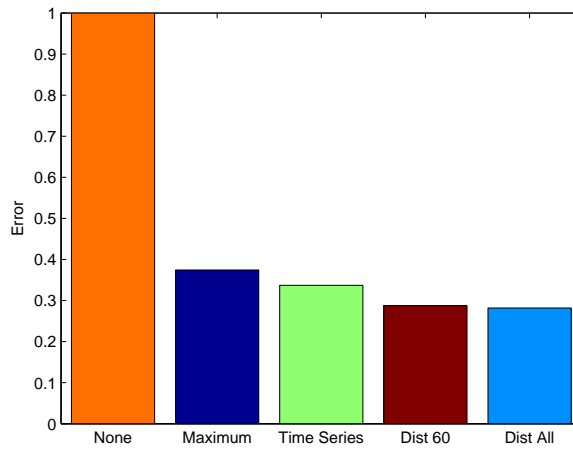


Figure 4.15: Errors of different methods (normalized)

We also use *CPC* to quantify the errors of the ensemble time series prediction and two best distribution based prediction method: *Dist 60* and *Dist All*. To better demonstrate the effectiveness of the prediction methods, we also calculate the cost of two naive method: *None* and *Maximum*. The method *None* takes no action for capacity planning. All the requests are responded by preparing the resource on-the-fly. The method *Maximum* always prepares the maximum capacity for the cloud. Figure 4.15 shows the evaluation result after normalization. This figure clearly shows that all the methods that take the pre-action are significantly better than *None*, which

does nothing. Moreover, all three sophisticated methods are better than *Maximum*. Among all these methods, *Dist All* makes least error among all the methods.

4.5.4 Type-aware Prediction Evaluation

To evaluate the effectiveness of request prediction for individual VM type, we compare the prediction performance of individual predictors with our ensemble predictor. Besides *CPC*, we also use *MAP*, *MSE* and *MAPE* to measure the precision of these predictors.

For VM type, we pick the top 12 (before the inflection point in Figure 4.12) most frequent VM types for experiments. All the time series are partitioned into two sets, the records for the last 30 days are used as the test data, while the remaining are used as the training data. Similar to the experiment in provisioning evaluation, grid search is utilized to seek the best parameter combinations for individual predictors.

Table 4.5 shows the precision of all the predictors on all the time series. Due to the space limit, we only list the details of the top 3 time series (*Red Hat Enterprise Linux 5.5 32-bit*, *Red Hat Enterprise Linux 5.5 64-bit*, and *SUSE Linux Enterprise Server*) and the average *CPC* of all the time series. It can be easily observed that the best predictor is different for different VM types. For example, GEP performs the best on the 1st VM type; ANN achieves good results on the 2nd VM type. Moreover, the winner predictor of one VM type can also perform badly for other VM types. For example, ANN obtains a poor precision on the 1st VM type.

For our ensemble predictor, although it does not perform the best on any single VM type, it is very robust as its performance is always close to the winner predictor on all the types. The average *CPC* shows that our ensemble predictor has the best average performance, indicating its self-adaptation capability.

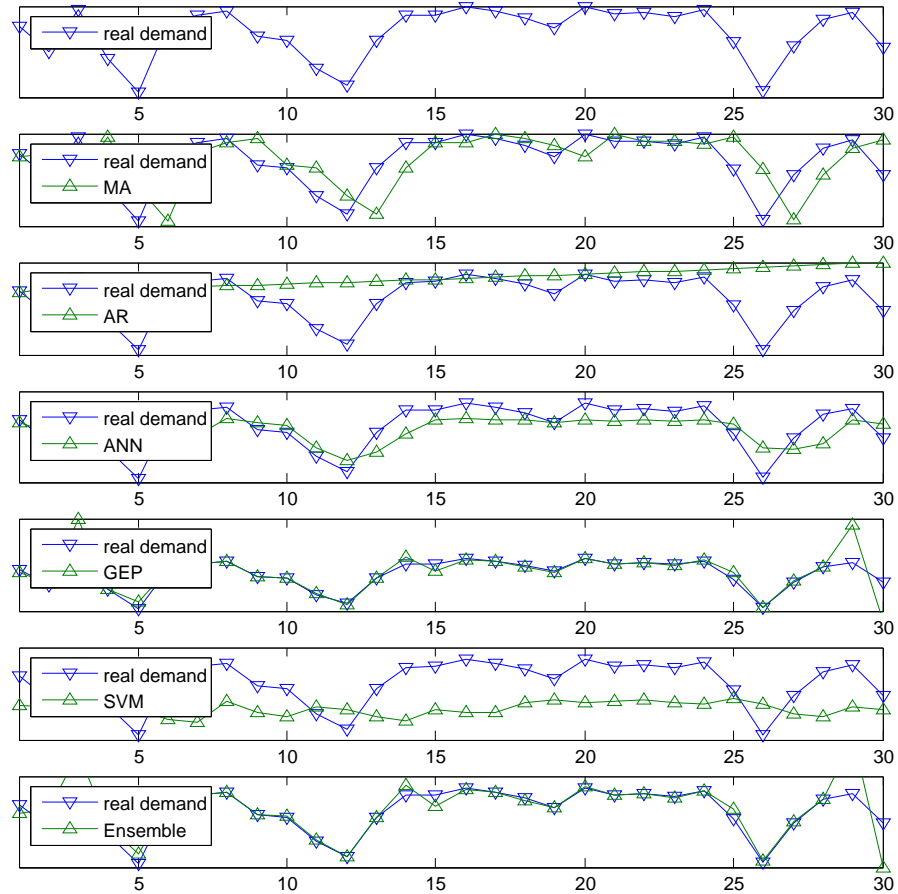


Figure 4.16: Prediction result of Red Hat Enterprise Linux 5.5 (32-bit)

Figure 4.16, 4.17, and 4.18 display the real time series of the three most frequent VM types and their corresponding prediction results of all the predictors. In these figures, ∇ denotes the original time series while \triangle denotes the predicted time series. From top to bottom, the time series are: (1) The real time series; (2) Time series predicted by MA; (3) Time series predicted by AR; (4) Time series predicted by ANN; (5) Time series predicted by GEP; (6) Time series predicted by SVM; (7) Time series predicted by Ensemble.

It is not difficult to observe that in all of these figures, the ensemble predictor can always identify the best predictor for the time series and quickly converge to it. Since under-prediction is worse than over-prediction in cloud provision scenario, the predictor that rarely under-predicts the demands is considered better than the one

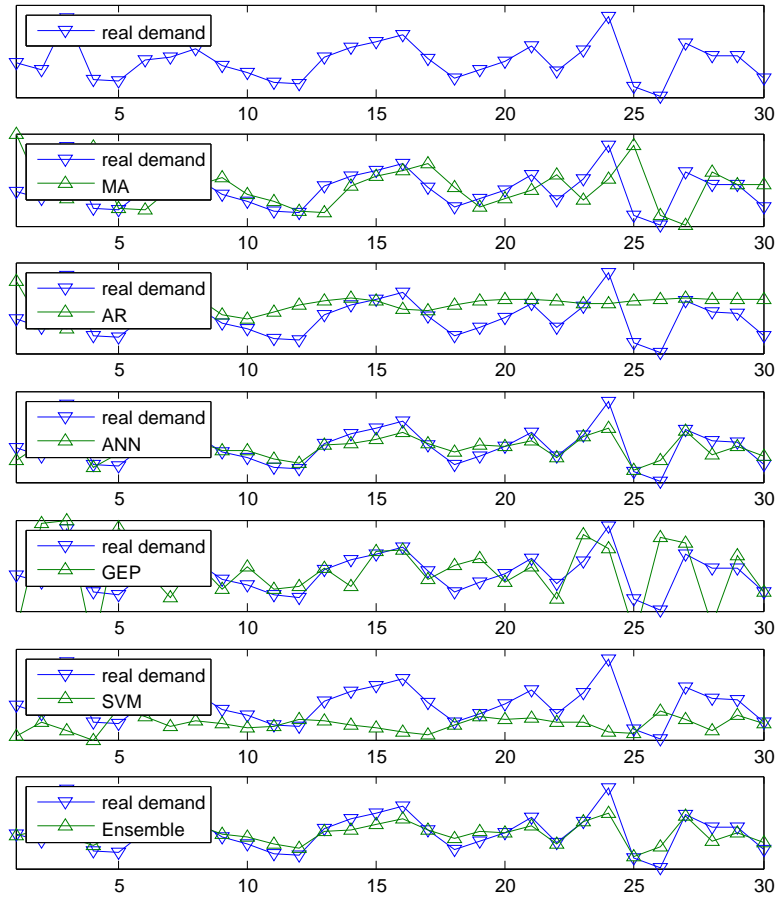


Figure 4.17: Prediction result of Red Hat Enterprise Linux 5.5 (64-bit)

whose outputs are always close to but less than the real demands. It can also be noted that although MA and SVM do not have the best performance in either of the three VM types, they can also make contributions to the ensemble predictor according to their weights.

Detailed Cost

To better investigate the composition of cost, we also calculate the prediction cost for each component.

Provisioning time reduction: The most important criterion to evaluate the performance is how much provisioning time can be saved. We calculate the proportion

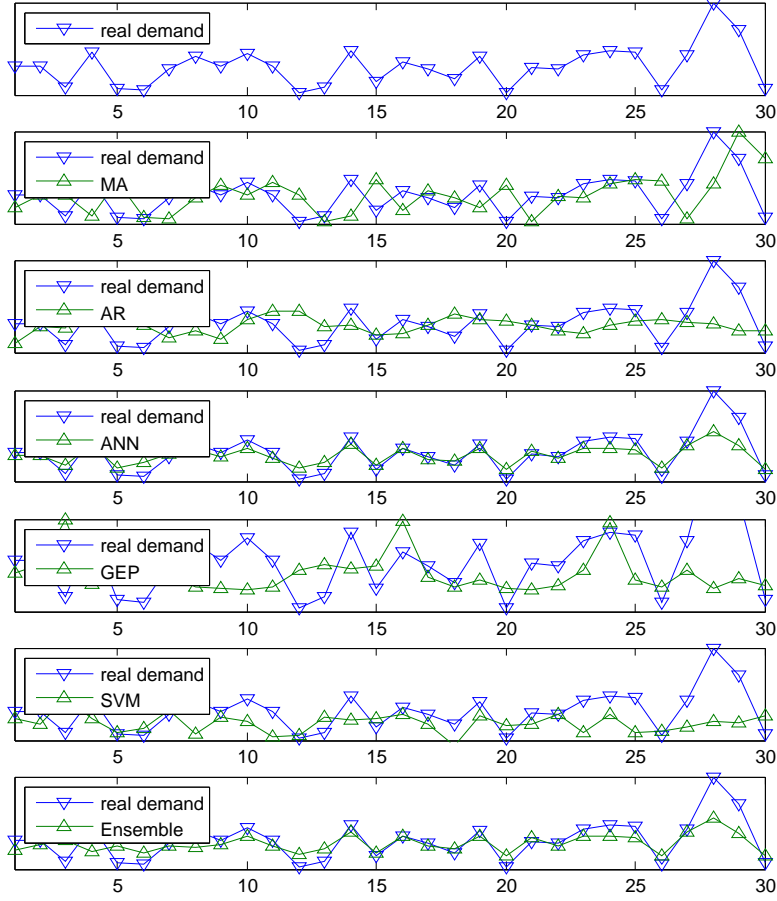


Figure 4.18: Prediction result SUSE Linux Enterprise Server

of time reduction obtained by predictors based on (4.12), where the save portion (p_{save}) is calculated according to Equation 4.12:

$$p_{save} = \frac{\sum_t (\max(v^{(t)} - \hat{v}^{(t)}, 0) T_{miss} + \hat{v}^{(t)} T_{hit})}{\sum_t v^{(t)} T_{miss}}. \quad (4.12)$$

Figure 4.19 shows the proportion of time reduction of each predictor. It is good to observe that most of the predictors can significantly decrease more than 60% of the provisioning time. However, in the presence of large variations across time series, the saved time achieved by the predictor is not stable for different VM types. On average, our ensemble predictor performs the best due to its strong self-adaptation ability.

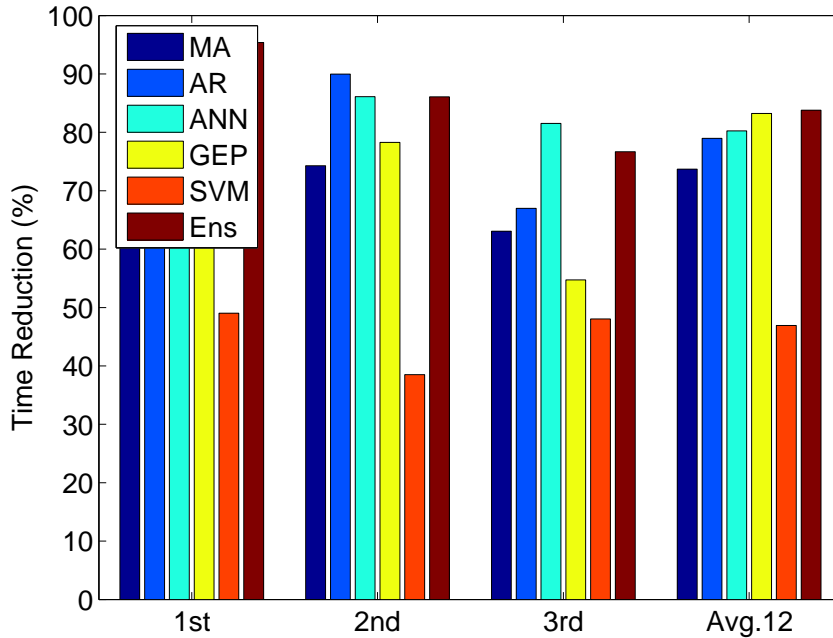


Figure 4.19: Average provisioning time reduction

Idled VMs

The cost of idled resources is another evaluation criterion of the quality of prediction. It is true that an always over-predicted predictor can save a lot of provisioning time, but such a predictor would also waste a lot of resources. Figure 4.20 shows the amount of idled resource caused by each predictor. On average, the best resource saver is SVM, but its performance in time reduction is the worst. Also note that GEP achieves a good performance on time reduction, but it wastes the resources twice as much as our method.

Effectiveness of Reservation Controller

Table 4.6 shows the ratio of provisioning time reduction that can be achieved by incorporating the *Reservation Controller*. For all the time series, *Reservation Controller* further improves the reduction of the average provisioning time from 83.79% to

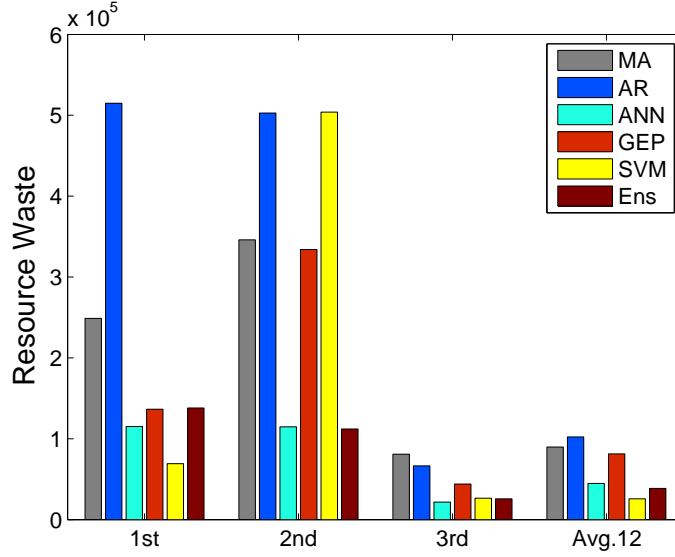


Figure 4.20: Average resource waste

94.22%. Moreover, with the assistance of reservation controller, the over-prediction portion of the VMs prepared before can be used for following days without going through the provisioning process again.

VM Type	1	2	3	4	5	6
Reduction	97.77%	91.58%	96.81%	96.26%	96.76%	91.67%
VM Type	7	8	9	10	11	12
Reduction	98.17%	94.37%	99.56%	85.35%	84.98%	97.45%

Table 4.6: Provisioning time Reduction by incorporating Reservation Controller

Effect of Sophisticated Cost Functions

As mentioned before, our proposed framework is flexible and can use different cost functions to guide the prediction process. In this section, we explore how the prediction is influenced by the use of complex cost functions.

For instance, in practice, a fixed amount of resource R_{fix} , i.e. the standing resources, is always provided for VMs preparation. If the over-predicted value lies below

such a threshold, the cost of resource is still 0. Equation 4.13 shows the form of this cost function:

$$R(v^{(t)}, \hat{v}^{(t)}) = \begin{cases} 0 & \text{if } \hat{v}^{(t)} < \min(R_{fix}, v^{(t)}), \\ (\hat{v}^{(t)} - R_{fix})R_{vm} & \text{if } v^{(t)} < R_{fix} < \hat{v}^{(t)}, \\ (\hat{v}^{(t)} - v^{(t)})R_{vm} & \text{if } R_{fix} \leq v^{(t)} < \hat{v}^{(t)}. \end{cases} \quad (4.13)$$

Figure 4.21 shows three predicted demands time series time generated by different R_{fix} values. As R_{fix} increases, the predicted demand tends to be more optimistic than pessimistic. Experimental results show that the average waiting time can be reduced by 93.94% when R_{fix} increases to 200, but the amount of idled resources becomes huge.

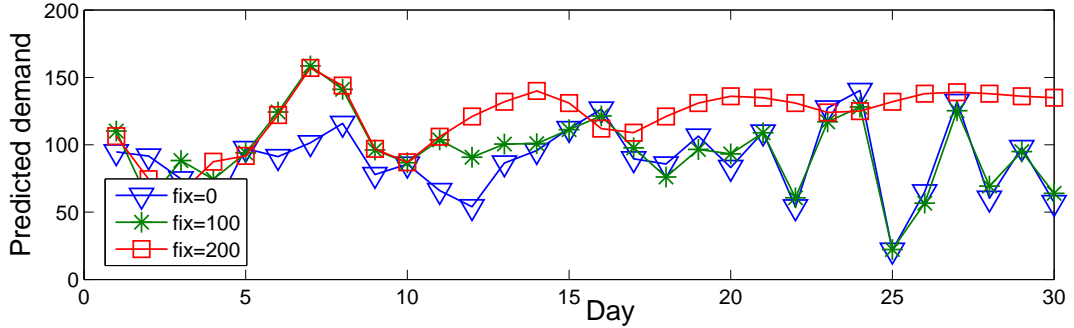


Figure 4.21: The influence of R_{fix} on prediction (normalized data)

4.5.5 Comparison of Different Measures

Table 4.7 shows the time reduction and the idled resources of the ensemble predictor guided by various cost measures. *CPC* clearly outperforms the other three cost measures in provisioning time reduction. We also find that *CPC* has the largest idled resources. Such a phenomenon can be well interpreted by the basic idea of these cost measures. *MAE*, *MSE* and *MAPE* are all symmetric cost measurements and

they guide the ensemble predictor to equally weight the under-predictors and over-predictors. While *CPC* gives more penalty to under-predictors than over-predictors, the ensemble predictor always prefers to giving larger weights to the over-predictor, which results in more time reduction and also more likely to waste resources. As in cloud service scenario, customer service quality is much more important, it is worth reducing the provisioning time on the costs of a reasonable amount of idled resources.

Time Reduction	Avg. Time Reduction	Avg. Resource Waste
MAE	81.77%	28300
MSE	80.13%	37100
MAPE	79.01%	33133
CPC	83.79%	38533

Table 4.7: Waiting Time Reduction and Resource Waste of Predictor Guided by Different Cost Measurements

4.5.6 Model Computational Cost

Our proposed method is neither computationally intensive nor storage intensive. In terms of CPU cost, given the historical requests (about tens of thousands per month), the training time costs less than 1 minute. Once the training is finished, the prediction can be conducted constantly (within 1 second).

The consumption of memory cost can also be ignored. This is because the only thing need to be put in memory is the parameters of these algorithms. For example, the number of parameters in *Neural Network* is $\sum_{i=1}^L s_{i-1} * s_i$, where s_i denotes the number of neurons in level i . Therefore, the total storage cost of the proposed method is no more than 1000.

In terms of scalability, the growth of cloud would only increase the value of requests at each timestamp, it would not increase the scale of the input data of the proposed algorithm.

4.6 Chapter Summary

In this chapter, we discussed two data mining based analytical techniques to improve the cloud service quality. Specifically, we believe that both the capacity planning and the instant VM provisioning problems can be handled by prediction, where the first problem can be solved by preparing the available resources beforehand and the second problem can be solved by pre-provisioning the needed VM instances. According to the unique characteristic of cloud service, we propose a novel cost-sensitive measure called *CPC* to guide the prediction procedure. To demonstrate the effectiveness of our approach, we implemented a prototype and conduct a series of simulation experiments based on the trace data of IBM Smart Cloud Enterprise. The experimental evaluation results demonstrated that our approach is able to effectively improve the service quality while retaining a low resource cost.

Temporal Mining for Stream Anomaly Detection

In this chapter, we will address the problem of timely and accurate anomaly identification problem in a distributed system. During my Ph.D study, part of the following content has been published.

In the following, the background and motivation of the stream anomaly detection will be introduced in Section 5.1. In Section 5.2, the stream anomaly detection problem will be formulated. In Section 5.3, the proposed solution framework and the detail solution will be presented. Finally, the summary of this chapter will be given in Section 5.6.

5.1 Background and Motivation

Anomaly detection has always been a critical and challenging problem in many applications. In the Big Data era, real-time processing is often needed by the applications. However, existing data processing infrastructures are designed based on inherent non-stream programming paradigm such as MapReduce [DG08], Bulk Synchronous Parallel (BSP) [Val90], and their variations. To reduce the processing delay, these applications have to be migrated to the stream processing engines [ABB⁺03, CCD⁺03]. Once the infrastructures are changed, the new data characteristics and analysis requirements make existing anomaly detection solutions no longer suitable.

5.1.1 A Motivating Example

Complex Event Processing (CEP) [ADGI08, MZZ12] is one quick solution to address the stream based anomaly detection issue. By expressing the anomalies detection rules with corresponding continuous query statements. This rule-based detection method can be applied to the scenarios where the anomaly can be clearly defined.

Besides CEP, several stream based anomaly detection algorithms have also been proposed. They either focus on identifying contextual anomaly over a collection of stable streams [BCFL09] or collective anomaly from one stream [AF07,PLL07]. These existing methods are useful in many applications but they are still unable pinpoint certain types of anomalies. Example 5 gives a simple example of such scenario.

Example 5. *Figure 5.1 illustrates the scenario of monitoring a 6-node computer cluster, where the x-axis denotes the time and the y-axis denotes the CPU utilization. The cluster has been monitored during time $[0, t_6]$. At time t_2 , a computing task has been submitted to the cluster and the cluster finishes this task at time t_4 . As is shown, two nodes (marked in dashed line) behave differently from the majority during some specific time periods. Node ① has a high CPU utilization during $[t_1, t_2]$ and a low CPU utilization during $[t_3, t_4]$ while node ② has a medium CPU utilization all the time. These two nodes with their associated abnormal periods are regarded as anomalies. Besides these two obvious anomalies, there are a slight delay on node ③ due to the network delay and a transient fluctuation on node ④ due to some random factors. However, they are normal phenomena in distributed systems and are not regarded as anomalies.*

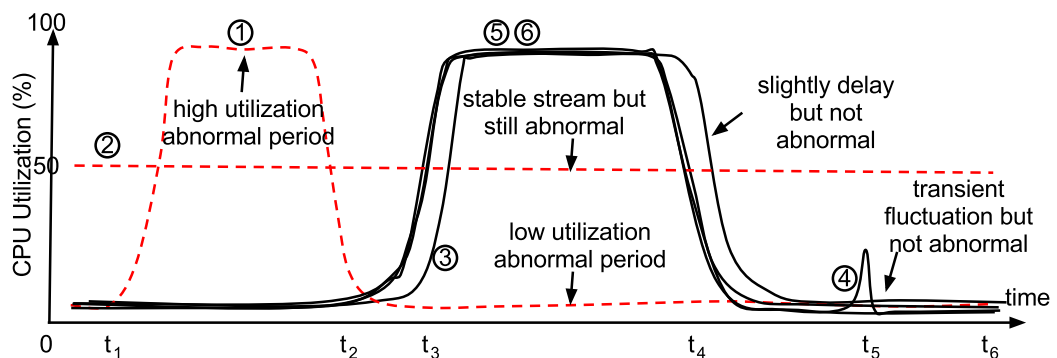


Figure 5.1: CPU utilization of a computing cluster

Figure 5.2 plots the ground truth as well as all the anomalies identified by existing methods including CEP queries with three different rules (Rule-CQ1, 2, and

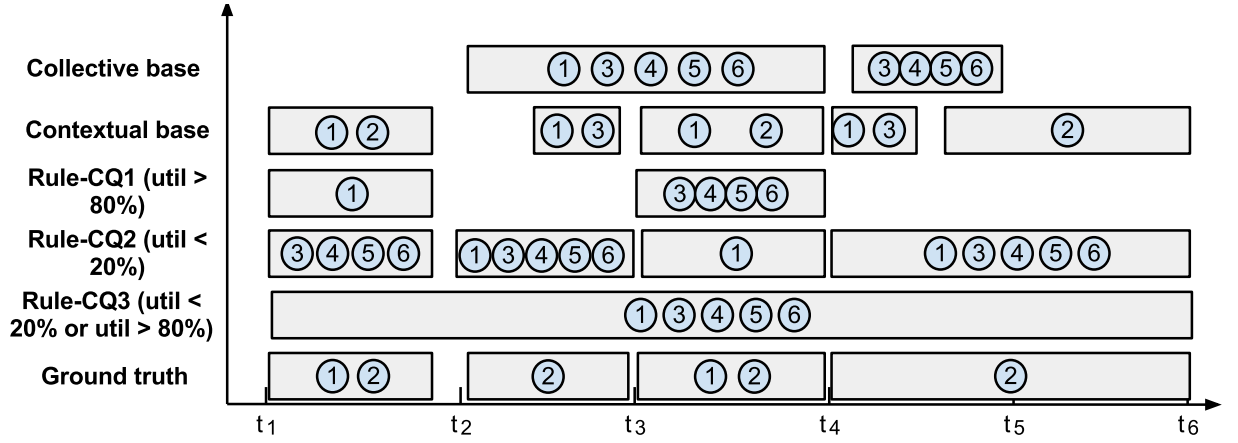


Figure 5.2: Identified anomalies in Example 5 (The box lists the IDs of abnormal streams during specified time period)

3), the collective based anomaly detection [BKNS00], and contextual based anomaly detection [CBK09].

To detect the anomalies via CEP query, the idea is to capture the events when the CPU utilizations of nodes are too high or too low. An example query following the syntax of [ADGI08] can be written as follows:

```
PATTERN SEQ(Observation o[])
WHERE avg(o[].cpu) oper threshold
(AND/OR avg(o[].cpu) oper threshold)*
WITHIN {length of sliding window}
```

where the selection condition in WHERE clause is the conjunction of one or more boolean expressions, *oper* is one of $\{>, <, <>, ==\}$, and *threshold* can be replaced by any valid expression. However, CEP queries are unable to correctly identify the anomalies in Figure 5.1 no matter how the selection conditions are specified. For instance, setting the condition as $\text{avg}(o[].\text{cpu}) > \{\text{threshold}\}$ would miss the anomalies during $[t_3, t_4]$ (Rule-CQ1); setting the condition as $\text{avg}(o[].\text{cpu}) < \{\text{threshold}\}$ would miss the anomalies during $[t_1, t_2]$ (Rule-CQ2); and combining the above two expressions with OR (Rule-CQ3) still cannot lead to the correct result. Besides deciding

the selection condition, how to rule out the situations of slight delays and transient fluctuations, and how to set the length of the sliding windows are all difficult problems when writing the continuous queries. The main reason is that the continuous query statement is not suitable to capture the contextual information where the “normal” behaviors are also dynamic (the utilizations of normal nodes also change over time in Figure 5.1).

Compared with CEP based methods, contextual anomaly detection methods achieve a better accuracy as they utilize the contextual information of all the streams. However, one limitation of contextual based methods is that they do not leverage the temporal information of streams and are not suitable for anomaly detection in dynamic environments. Therefore, these methods would wrongly identify the slightly delayed and fluctuated nodes as anomalies.

For the given example, collective anomaly detection methods do not work well neither. This is because these methods would identify the anomaly of each stream based on its normal behaviors. Once the current behavior of a stream is different from its normal behavior (inferred based on historical data), it is considered as abnormal. In the example, when the cluster works on the task during $[t_3, t_4]$, all the working nodes would be identified as abnormal due to the sudden burst.

5.2 Problem Statement

In this section, the notations and definitions that are relevant to the anomaly detection problem will be given first. Then, the problem based on the given notations and definitions will be formally defined.

Definition 5.2.1. DATA STREAM. *A data stream S_i is an ordered infinite sequence of data instances $\{s_{i1}, s_{i2}, s_{i3}, \dots\}$. Each data instance s_{it} is the observation of data stream S_i at timestamp t .*

The data instances s_{it} in S_i can have any number of dimensions, depending on the concrete applications. For the remaining of this paper, the terms “data instance” and “observation” would be used interchangeably. To make the notation uncluttered, we use s_i in places where the absence of timestamp do not cause the ambiguity.

Definition 5.2.2. STREAM COLLECTION. *A stream collection $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$ is a collection of n unordered data streams.*

The input of our anomaly detection framework is a *stream collection*. For instance, in example 5, the input stream collection is $\mathcal{S} = \{\textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}, \textcircled{5}, \textcircled{6}\}$

Definition 5.2.3. SNAPSHOT. *A snapshot is a set of key-value pairs $S^{(t)} = \{S_i : s_{it} | S_i \in \mathcal{S}\}$, denoting the set of the observations $\{s_{1t}, s_{2t}, \dots, s_{|\mathcal{S}|t}\}$ of the data streams in stream collection \mathcal{S} at time t .*

A *snapshot* captures the configuration of the stream collection for a certain moment. Taking Figure 5.1 for example, the snapshot at time t_5 is $S^{(t_5)} = \{\textcircled{1} : 0\%, \textcircled{2} : 50\%, \textcircled{3} : 0\%, \textcircled{4} : 20\%, \textcircled{5} : 0\%, \textcircled{6} : 0\%\}$. For simplicity, we use $S(i)$ to denote the i th dimension of the observations in a certain snapshot.

Definition 5.2.4. CONTEXTUAL COLLECTIVE ANOMALY. *A contextual collective stream anomaly is denoted as a tuple $\langle S_i, [t_b, t_e], N \rangle$, where S_i is the ID of a single data stream from the collection of data streams \mathcal{S} , $[t_b, t_e]$ is the associated time period when S_i is observed to constantly deviate from the majority streams in \mathcal{S} , and N indicates the severity of the anomaly.*

In Example 5, 3 contextual collective anomalies can be found in total. During time period $[t_1, t_2]$, node $\textcircled{1}$ behaves constantly different from the other nodes, so there is an anomaly $\langle \textcircled{1}, [t_1, t_2], N_1 \rangle$. The other two contextual collective anomalies, $\langle \textcircled{1}, [t_3, t_4], N_2 \rangle$ and $\langle \textcircled{2}, [0, t_6], N_3 \rangle$, can also be found with the same reason.

In Definition 5.2.4, the severity of deviation is measured in a given metric space \mathcal{M} with a distance function $f : s_{it} \times s_{ku} \rightarrow \mathbb{R}$. For simplicity, we use *Euclidean Distance* as an example throughout this paper.

Problem Definition. The anomaly detection problem in our paper can be described below: Given a stream collection $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$, identify the source of the contextual collective anomalies S_i , the associated time period $[t_s, t_e]$, as well as a quantification about the confidence of the detection p . Moreover, the detection has to be conducted on data streams that look-back is not allowed and the anomalies need to be identified in real time.

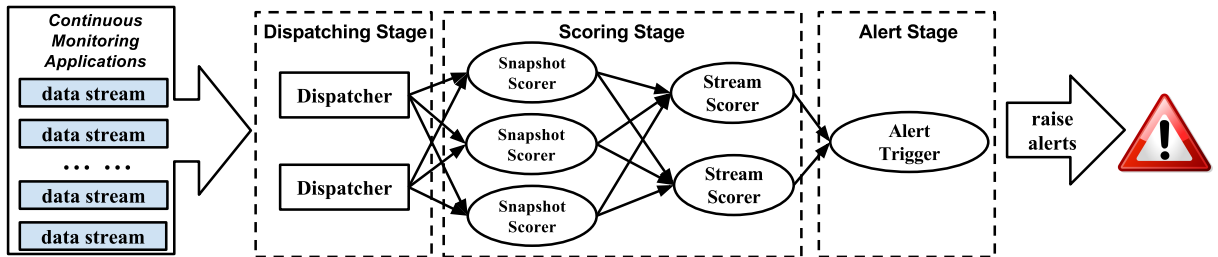


Figure 5.3: Distributed real time stream anomaly detection framework

5.3 Framework Overview

In this section, we briefly describe how the aforementioned problem is addressed and then introduce the proposed distributed real time anomaly detection framework from a high level perspective.

As previously mentioned, this paper focuses on discovering contextual collective anomalies over a collection of data streams obtained from a homogeneous distributed environment. An example of the homogeneous distributed environment is the system with load balance, which is widely used at the backend by the popular web sites like Google and Facebook.

It is known that, in such a kind of environment, the components should behave similar to each other. Therefore, the snapshots (the current observation) of these streams should be close to each other at any time. Naturally, we need to identify the anomalies by investigating both contextual information (the information of the current snapshot) and collective information (the historical information).

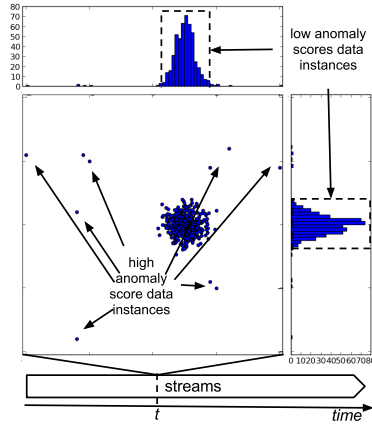


Figure 5.4: The snapshot at a certain timestamp

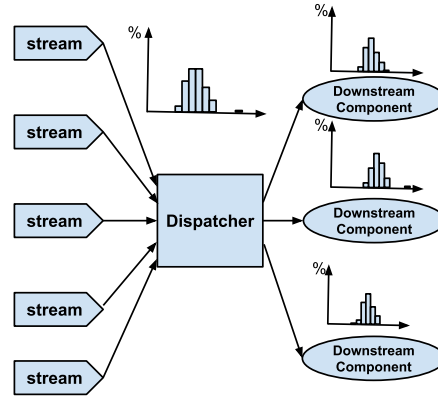


Figure 5.5: Random partition of data instance

Figure 5.3 illustrates our proposed framework for contextual collective anomaly detection. The anomaly detection is conducted in three stages: the *dispatching stage*, the *scoring stage*, and the *alert stage*. The functionality of the three stages are briefly described as follows:

- *Dispatching stage*: This stage uses *dispatchers* to receive the observations from external data sources and then shuffle the observations to different downstream processing components.
- *Scoring stage*: This stage quantifies the candidate anomalies first using *snapshot scorer* then *stream scorer*.

The *snapshot scorer* leverages contextual information to quantify the confidence of anomaly for each data instance at a given *snapshot*. Taking Figure 5.4 for example, it shows the data distribution by taking the snapshot of the

2-dimensional data instances of 500 streams at timestamp t . As shown, most of the data instances are close to each other and located in a dense area. These data instances are not likely to be identified as anomalies as their instance anomaly scores are small. On the contrary, a small portion of the data instances (those points that are far away from the dense region) have larger instance anomaly scores and are more likely to be abnormal.

A data instance with a high anomaly score does not indisputably indicate its corresponding stream to be a real anomaly. This is because the transient fluctuation and phase shift are common in real world distributed environment. To mitigate such effects, the *stream scorer* is designed to handle the problem. In particular, the *stream scorer* combines the information obtained from the *instance scorer* and the historical information of each stream to quantify the anomaly confidence of each stream.

- *Alert stage*: The alert stage contains the *alter trigger*. The alert triggers leverages the unsupervised learning methods to identify and then reports the outliers.

The advantage of our framework is reflected by the ease of integration, the flexibility, and the algorithm independence. Firstly, any external data sources can be easily feed to the framework for anomaly detection. Moreover, the components in every stage can be scaled-out to increase the processing capability if necessary. The number of components in each stage can be easily customized according to the data scale of concrete applications. Furthermore, the algorithms in each stage can be replaced and upgraded with better alternatives and the replacement would not interfere with other stages.

5.4 Stream Anomaly Detection Methodology

5.4.1 Data Receiving and Dispatching

The dispatching stage is an auxiliary stage in our framework. When the data scale (i.e., the number of distinct data streams) is too large for a single computing component to process within in a reasonable time delay, the dispatcher would shuffle the received observations to downstream computing components before the scoring stage (as shown in Figure 5.5). By leveraging random shuffling algorithm like Fisher-Yates shuffle [FY⁺49], dispatching can be conducted in constant time per observation. After dispatching, each downstream component would conduct scoring independently and parallel on a sampled stream observations with identical distribution. As the observations are randomly sampled, the performance of the anomaly detection will not be affected.

Ideally, the observations coming from the homogeneous data sources have very similar measurable value (e.g. workload of each server in a load balanced system), but some unknown random factors can easily cause the variations of the actual observations. Therefore in fact, an observation $\mathbf{s}_i \in \mathbb{R}^d$ is viewed as the ideal case value \mathbf{s}_{ideal} with additive Gaussian noise so that $\mathbf{s}_i = \mathbf{s}_{ideal} + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. For those data sources in abnormal conditions, their observations are generated in a slightly different but unknown way. Suppose N_i is the number of observations in partition i , it's not difficult to know that, given enough observations, the mean and covariance can be easily estimated locally using maximum likelihood estimation, i.e. $\hat{\boldsymbol{\mu}}_{ML} = \frac{\sum_i \mathbf{s}_i}{N_i}$ and $\widehat{cov}(S(x), S(y))_{ML} = \frac{1}{n-1} \sum_{i=1}^n (s_i(x) - \bar{S}(x))(s_i(y) - \bar{S}(y))$, where $\bar{S}(x)$ and $\bar{S}(y)$ respectively denote the sample mean of dimension x and y .

5.4.2 Snapshot Anomaly Quantification

Quantifying the anomaly in a snapshot is the first task in the *scoring stage*. For doing so, we leverage *snapshot scorer* in this step. This score measures the amount of deviation of the specified observation s_{it} to the center of all the observations in a snapshot at timestamp t .

A common solution to quantify the anomaly score of multi-dimension data is to use Local Outlier Factor (LOF) [BKNS00]. In principle, LOF measures the anomaly score by making use of the density-based clustering. This method is useful for offline mining but is not suitable in our scenario due to the following two limitations: (1) LOF is not aware of the scale inconsistency among different dimensions. For dimensions with inconsistent scales, LOF would be dominated by the dimensions with large scales. Taking the system monitoring application for example, the CPU utilization is represented by the percentage of clock ticks used per second, while the memory usage is represented by the amount of RAM in KB, MB or GB, for which the later has a much larger scale. When using LOF, the dimension of memory usage would dominate the anomaly score. (2) The time of computing LOF score of observations increases superlinearly ($O(dn \log |S|)$) as the number of observations increases. This time complexity is acceptable for offline detection but is prohibitive for real time detection.

To address the above two limitations, we propose a simple yet efficient method to quantify the data instance anomaly scores. The basic idea is that the anomaly score of an observation is quantified as the amount of uncertainty it brings to the snapshot $S^{(t)}$. As the observations in a snapshot follows the normal distribution, it is suitable to use the increase of entropy to measure the anomaly of an observation. To quantify the anomaly score, two types of variance are needed: the *variance* and

the *leave-one-out variance*, where the leave-one-out variance is the variance of the distribution when one specific data instance is not counted.

Algorithm 3 Snapshot Anomaly Quantification

1. **INPUT:** Snapshot $S^{(t)} = (s_1, \dots, s_n)$, for $s_i \in \mathbb{R}^d$.
 2. **OUTPUT:** Snapshot anomaly scores $N \in \mathbb{R}^{|S|}$.
 3. Create a $d \times |S|$ matrix $M = (s_1, \dots, s_n)$
 4. Conduct 0-1 normalization on rows of M .
 5. $\mathbf{x} = \mathbf{0}^d$ and $N = \mathbf{0}^d$
 6. $\mathbf{m} = (\mathbb{E}(S(1)), \dots, \mathbb{E}(S(d)))^T$
 7. $M = (s_{1t} - \mathbf{m}, s_{2t} - \mathbf{m}, \dots, s_{dt} - \mathbf{m})$
 8. **for** $j = 0 \rightarrow k$ **do**
 9. $\mathbf{x}_j = \|\text{jth column of } M\|_2^2$
 10. **end for**
 11. **for all** $s_i \in S^{(t)}$ **do**
 12. Calculate N_i according to Equation (5.1).
 13. **end for**
 14. Conduct 0-1 normalization on N .
 15. **return** N
-

A naive algorithm to quantify the anomaly scores requires quadratic time ($O(d|S| + d|S|^2)$). By reusing the intermediate results, we propose an improved algorithm with time complexity linear to the number of streams. The pseudo code of the proposed algorithm is shown in Algorithm 3. As illustrated, matrix M is used to store the distances between each dimension of the observations to the corresponding mean. Making use of M , the *leave-one-out variance* can be quickly calculated as $\sigma_{ik} = \frac{n\sigma_k - M(i,j)}{n-1}$, where σ_k denotes the variance of dimension k and σ_{ik} denotes the leave-one-out variance of dimension k by excluding s_i . As the entropy of normal distribution is $H = \frac{1}{2} \ln(2\pi e\sigma^2)$, the increase of entropy for observation s_i at dimension k can be calculated as

$$d_k = H'_k - H_k = \frac{1}{2} \ln \frac{\sigma_{ik}}{\sigma_k} = \frac{1}{2} \ln \frac{(\mathbf{x}_j - M_{ij}^2)/(n-1)}{\mathbf{x}_j/n} \quad (5.1)$$

Summing up all dimensions, the snapshot anomaly score of s_{it} is $N_{it} = \sum_k d_k$. Note that the computation implicitly ignores the correlation between dimensions.

This is because if an observation is an outlier, the correlation effect would only deviate it further from other observations.

5.4.3 Stream Anomaly Quantification

As a stream is continuously evolving and its observations only reflect the transient behavior, simply using snapshot anomaly score alone for anomaly detection would result in a lot of false-positives due to the transient fluctuation and the slight phase shift phenomenon. To mitigate such situations, it is needed to quantify the severity of anomaly by using the stream anomaly scores that make use of the historical information of the stream.

An intuitive way to solve this problem is to calculate the stream anomaly score from the recent historical instances stored in a sliding window. However, this solution has two obvious limitations: (1) It is hard to decide the window length. A long sliding window would miss the real anomaly while a short sliding window cannot rule out the false-positives. (2) It ignores the impact of observations that are not in the sliding window. The observation that is just popped out from the sliding window would immediately and totally lose its impact to the stream.

To well balance the history and the current observation, we use *stream anomaly score* N_i to quantify how significant a stream S_i behaves differently from the majority of the streams. To quantify N_i , we exploit the exponential decay function to control the influence depreciation. Supposing Δt is the time gap between two adjacent observations, *the influence of an observation s_{it} at timestamp $t_{x+k} = t_x + k\Delta t$* can be expressed as $N_{it_x}(t_{x+k}) = N_{it_x}(t_x + k\Delta t) = N_{it_x}e^{-\lambda kt}$, ($\lambda > 0$), where λ is a parameter to control the decay speed. In the experiment evaluation, we will discuss how this parameter affects the anomaly detection results.

To make the notation uncluttered, we use t_{-i} to denote the timestamp that is $i\Delta t$ ahead of current timestamp t , i.e. $t_{-i} = t - i\Delta t$. Summing up the influences of all the historical observations, the overall historical influence I_{it} for current timestamp t can be expressed as Equation (5.2).

$$\begin{aligned}
I_{it} &= N_{it_{-1}}(t) + N_{it_{-2}}(t) + N_{it_{-3}}(t) + \dots \\
&= N_{it_{-1}}e^{-\lambda} + N_{it_{-2}}e^{-2\lambda} + N_{it_{-3}}e^{-3\lambda} + \dots \\
&= e^{-\lambda}(N_{it_{-1}} + e^{-\lambda}(N_{it_{-2}} + e^{-\lambda}(N_{it_{-3}} + \dots \\
&= e^{-\lambda}(N_{it_{-1}} + I_{it_{-1}}).
\end{aligned} \tag{5.2}$$

The stream anomaly score of stream S_i is the summation of the data instance anomaly score of current observation N_{it} and the overall historical influence, i.e.,

$$N_i = N_{it} + I_{it}. \tag{5.3}$$

As is shown in Equation (5.2), the overall historical influence can be incrementally updated with cost $O(1)$ for both time and space complexity. Therefore, *stream anomaly scorer* can be efficiently computed.

Properties of Stream Anomaly Score

The properties of stream anomaly score make our framework insensitive to the transient fluctuation and effective to capture the real anomaly.

Comparing to the transient fluctuation, the real anomaly is more durable. Figure 5.6 shows the situations of a transient fluctuation (in the left subfigure) and a real anomaly (in the right subfigure). In both situations, the stream behaves normally before timestamp t_x . For the left situation, a transient fluctuation occurs at timestamp t_{x+1} , and then the stream returns to normal at timestamp t_{x+2} . For the right situation, an anomaly begins at timestamp t_{x+1} , lasts for a while till timestamp t_{x+k} , and then the stream returns to normal afterwards. Based on Figure 5.6, we show two properties about the stream anomaly score.

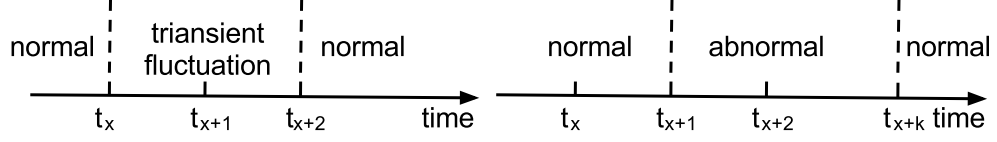


Figure 5.6: Transient fluctuation and anomaly

Property 1. *The increase of stream anomaly score caused by transient disturbance would decrease over time.*

Proof. Suppose a transient fluctuation occurs at timestamp t_{x+1} in stream S_i , in the worst case, the difference between the data instance scores of the stream S_i and a non-fluctuated stream S_j is at most $d_{upper} = N_{it_{x+1}} - N_{jt_{x+1}} \leq \sqrt{|d|}$, where $|d|$ is the number of dimensions.

Let $\delta_1 = I_{it_{x+1}} - I_{jt_{x+1}}$. Before timestamp t_{x+1} , no stream is abnormal, so $I_{it_{x+1}}$ and $I_{jt_{x+1}}$ are close enough and the expectation $E(\delta_1) = 0$. At timestamp t_{x+1} , a transient fluctuation occurs in stream S_i . According to Equation (5.3), the difference of the stream anomaly scores is

$$\begin{aligned} N_i - N_j &= (N_{it_{x+1}} + I_{it_{x+1}}) - (N_{jt_{x+1}} - I_{jt_{x+1}}) \\ &= N_{it_{x+1}} - N_{jt_{x+1}} + \delta_1. \end{aligned} \tag{5.4}$$

At timestamp t_{x+2} , S_i turns to be normal again, so $N_{it_{x+2}}$ equals to $N_{jt_{x+2}}$ on average. Let $\delta_2 = N_{it_{x+2}} - N_{jt_{x+2}}$, we can also get $E(\delta_2) = 0$.

Accordingly, the difference between the corresponding stream anomaly scores at timestamp $x + 2$ becomes

$$\begin{aligned}
N'_i - N'_j &= (N_{it_{x+2}} + I_{it_{x+2}}) - (N_{jt_{x+2}} + I_{it_{x+2}}) \\
&= I_{it_{x+2}} - I_{jt_{x+2}} + \delta_2 \\
&= e^{-\lambda}(N_{it_{x+1}} + I_{it_{x+1}}) - e^{-\lambda}(N_{jt_{x+1}} + I_{jt_{x+1}}) + \delta_2 \quad (5.5) \\
&= e^{-\lambda}(N_{it_{x+1}} - N_{jt_{x+1}} + \delta_1) + \delta_2 \\
&< (N_{it_{x+1}} - N_{jt_{x+1}} + \delta_1) + \delta_2 = N_i - N_j.
\end{aligned}$$

According to Equation (5.5), it is known that at timestamp t_{x+2} , the effect of fluctuation at timestamp t_{x+1} decreases. \square

Property 2. *The increase of stream anomaly score caused by anomaly would be accumulated over time.*

Proof. If a stream begins to be abnormal at timestamp t_{x+1} , its data instance scores would become larger than those of the normal streams during the abnormal period. Suppose ϵ is the difference of the data instance scores between the abnormal stream S_i and a normal stream S_j , at timestamp t_{x+1} , the difference of the stream anomaly scores is

$$\Delta = N_i - N_j = N_{it_{x+1}} - N_{jt_{x+1}} + (I_{it_{x+1}} - I_{jt_{x+1}}) = \epsilon.$$

In the above equation, since both streams S_i and S_j are normal before timestamp t_{x+1} , we have $\delta_1 = I_{it_{x+1}} - I_{jt_{x+1}}$ and the expectation $E(\delta_1) = 0$. At timestamp t_{x+2} , since the stream S_i is still in the abnormal period, the difference of data instance scores is still larger than or equal to ϵ , and the difference of stream anomaly scores between these two streams at timestamp t_{x+2} is

$$\begin{aligned}
\Delta' &= N'_i - N'_j = N_{it_{x+2}} - N_{jt_{x+2}} + (I_{it_{x+2}} - I_{jt_{x+2}}) \\
&\geq \epsilon + e^{-\lambda}((N_{it_{x+1}} + I_{it_{x+1}}) - (N_{jt_{x+1}} + I_{jt_{x+1}})) \quad (5.6) \\
&= \epsilon + e^{-\lambda}(N_i - N_j) > \epsilon.
\end{aligned}$$

According to Equation (5.6), we can conclude that once a stream becomes abnormal, the difference between its stream anomaly score and those of the normal streams would increase over time. \square

Similar properties can also be shown for the situation of slight shifts. A slight shift can be treated as two transient fluctuations occurring at the beginning and the end of the shift. In the next section, we will leverage these two properties to effectively identify the anomalies in the ALERT STAGE.

5.4.4 Alert Triggering

Most of the stream anomaly detection solutions [GXZ⁺10] identify the anomalies by picking the streams with top- k anomaly scores or the ones whose scores exceed a predefined threshold. However, these two approaches are not practical in real world applications for the following reasons: (1) *Threshold is hard to set.* It requires the users to understand the underlying mechanism of the application to correctly set the parameter. (2) *The number of anomalies are changing all the time.* It is possible that more than k anomaly streams exist at one time, then the top- k approach would miss these real anomalies.

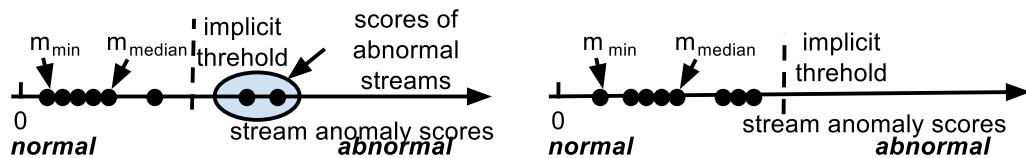


Figure 5.7: Abnormal streams identification

To eliminate the parameters, we propose an unsupervised method to identify and quantify the anomalies by leveraging the distribution of the anomaly scores. The first step is to find the median of the stream anomaly scores (N_{median}). If the distance between a stream anomaly score and the median score is larger than the distance

between the median score and the minimal score (N_{min}), the corresponding stream is regarded as abnormal. As shown in Figure 5.7, this method implicitly defines a dynamic threshold (shown as the dashed line) based on the hypothesis that there is no anomaly. If there is no anomaly, the skewness of the anomaly score distribution should be small and the median score should be close to the mean score. If the hypothesis is true, $N_{median} - N_{min}$ should be close to half of the distance between the minimum score and the maximum score. On the contrary, if a score N_i is larger than $2 \times (N_{median} - N_{min})$, the hypothesis is violated and all the streams with scores at least N_i are abnormal.

Besides the general case, we also need to handle one special case: a transient fluctuation occurs at the current timestamp. According to Property (1) in Section 5.4.3, the effect of transient fluctuation is at most $d_{upper} = N_{it_{x+1}} - N_{jt_{x+1}}$ and it will monotonically decrease. Therefore, even a stream whose anomaly score is larger than $2 \times (N_{median} - N_{min})$, it can still be a normal stream if the difference between its anomaly score and N_{min} is smaller than d_{upper} . To prune the false-positive situations caused by transient fluctuation, the stream is instead identified as abnormal if

$$N_i > \max(2(N_{median} - N_{min}), N_{min} + d_{upper}). \quad (5.7)$$

Another thing needs to be noted is that the stream anomaly scores have an upper bound $\frac{d_{upper}}{1-e^{-\lambda}}$. According to the property of convergent sequence, the stream anomaly scores of all streams would converge to this upper bound. When the values of stream anomaly scores are close to the upper bound, they tend to be close to each other and hard to be distinguished. To handle this problem, we reset all the stream anomaly scores to 0 whenever one of them close to the upper bound.

In terms of the time complexity, the abnormal streams can be found in $O(n)$ time. Algorithm 4 illustrates the algorithm of stream anomalies identification. The median of the scores can be found in $O(n)$ in the worst case using the *BFPRT* al-

Algorithm 4 Stream Anomaly Identification

1. **INPUT:** λ , and unordered stream profile list $\mathcal{S} = \{S_1, \dots, S_n\}$.
 2. $mIdx \leftarrow \lceil \frac{2}{n} \rceil$
 3. $N_{median} \leftarrow \text{BFPRT}(\mathcal{S}, mIdx)$
 4. $N_{min} \leftarrow \min(S_i.score | 0 \leq i \leq mIdx)$
 5. $N_{max} \leftarrow N_{median}$
 6. **for** $i \leftarrow mIdx$ to n **do**
 7. **if** Condition (5.7) is satisfied **then**
 8. Trigger alert for S_i with score N_i at current time.
 9. **if** $N_i > N_{max}$ **then**
 10. $N_{max} \leftarrow N_i$
 11. **end if**
 12. **end if**
 13. **end for**
 14. **if** N_{max} is close to the upper bound **then**
 15. Reset all stream anomaly scores.
 16. **end if**
-

gorithm [BFP⁺73]. Besides finding the median, this algorithm also partially sorts the list by moving smaller scores before the median and larger scores after the median, making it trivial to identify the abnormal streams by only checking the streams appearing after the median.

5.5 Experimental Evaluation

To investigate the effectiveness and efficiency of our framework, we design several sets of experiments with two real world data applications: *anomaly detection over a computing cluster* and *topic anomaly detection on twitter*. Taking these two applications as case studies, we show that our proposed framework can effectively identify the abnormal behavior of streams. It should be pointed out that our proposed framework can also be applied to many other application areas such as PM2.5 environment monitoring, healthcare monitoring, and stock market monitoring.

5.5.1 Real World Scenario — Anomaly Detection of Computing Cluster

System anomaly detection is one of the critical tasks in system management. In this set of experiments, we show that our proposed framework can effectively and efficiently discover the abnormal behaviors of the computer nodes with high precision and low latency.

Experiment Settings

For the experiments, we leverage a distributed system monitoring tool [ZJZ⁺13] into a 16-node computing cluster. Then we deploy the proposed anomaly detection program on an external computer to analyze the collected trace data in real time. To well evaluate our proposed framework, we terminate all the irrelevant processes running on these nodes. On this nodes, we intentionally inject various types of anomalies and monitor their running status for 1000 seconds. The source code of injection program is available at <https://github.com/yxjiang/system-noiser>. The details of the injections are listed in Table 5.1.

Table 5.1: List of Injections

No.	Time Period	Node	Description
1	[100, 150]	2	Keep CPU utilization above 95.
2	[300, 400]	3	Keep memory usage at 70%.
3	[350, 400]	3	Keep CPU utilization above 95%.
4	[600, 650]	4	Keep memory usage at 70%.
5	[900, 950]	2,5	Keep CPU utilization above 95%.
6	[800, 850]	1-5,7-16	Keep CPU utilization above 95%.

Through these injections, we can answer the following questions about our framework: (1) Whether our framework can identify the anomalies with different types of root causes. (2) Whether our framework can identify multiple anomalies occurring simultaneously.

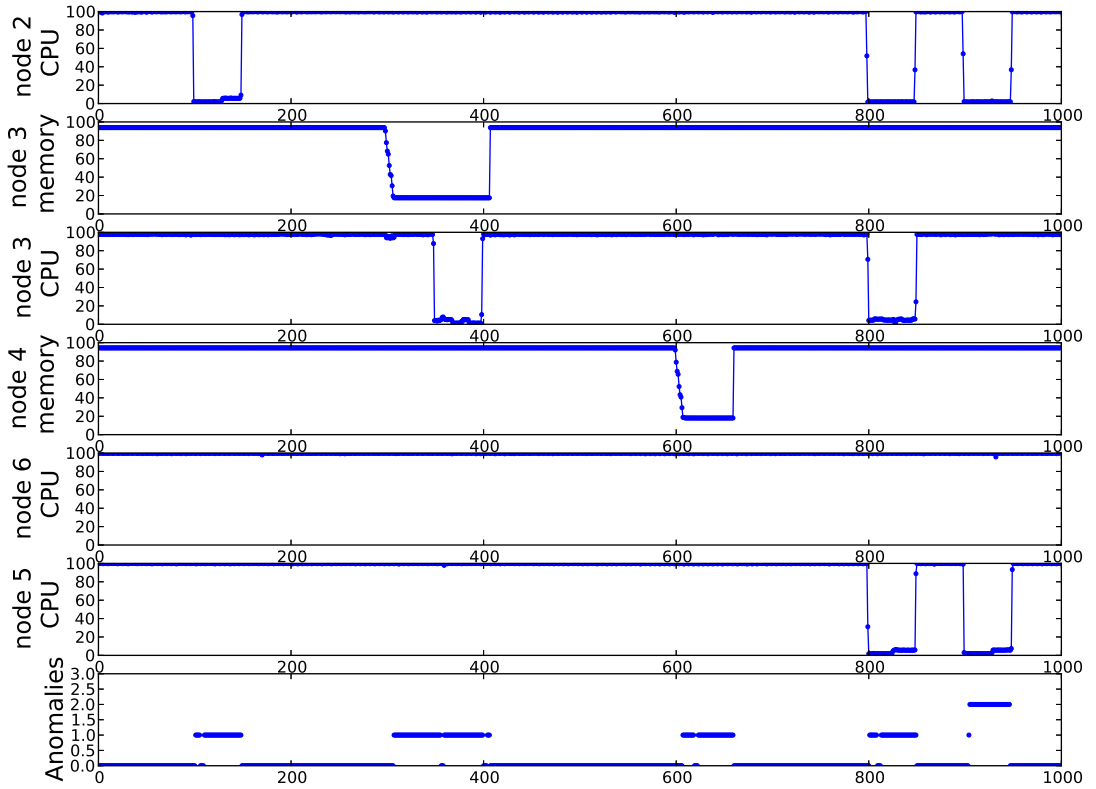


Figure 5.8: Injections and the captured alerts

Results Analysis

Figure 5.8 illustrates the results of this experiment by plotting the actual injections (top 6 sub-figures) as well as the captured alerts (the bottom subplots), where the x-axis represents the time and the y-axis represents the idled CPU utilization, idle memory usage or the number of anomalies in each timestamp. We evaluate the framework from 5 aspects through carefully-designed injections.

1. *Single dimension (e.g. idle CPU utilization or idle memory usage) of a single stream behaves abnormally.* This is the simplest type of anomalies. It is generated by injections No.1 and No.4 in Table 5.1. As shown in Figure 5.8, our framework effectively identifies these anomalies with the correct time periods.
2. *Multiple dimensions (e.g. CPU utilization and memory usage) of a single stream*

behaves abnormally at the same time. This type of anomalies is generated by injections No.2 and No.3 in Table 5.1, and our framework correctly captures such anomalies during the time period [300, 400]. One thing should be noted is that the stream anomaly score of node 3 increases faster during the time period [350, 400] than the time period [300, 350]. This is because two types of anomalies (CPU utilization and memory usage) appear simultaneously during the time period [350, 400].

3. *Multiple streams behave abnormally simultaneously.* This type of anomalies is generated by injection No.5. During the injection time period, our framework correctly identifies both anomalies (on node 2 and node 5).
4. *Stable but abnormal streams.* This kind of anomaly is indirectly generated by injection No.6 in Table 5.1. This injection emulates the scenario that all the nodes but one (i.e., node 6) in a cluster received the command of executing a task. As is shown, although the CPU utilization of node 6 behaves stable all the time, it is still considered to be abnormal during the time period [800, 850]. This is because it remains idle when all the other nodes are busy.
5. *Transient fluctuation and slight delay would not cause false-positive.* As this experiment is conducted in a distributed environment, delays exist and vary for different nodes when executing the injections. Despite this intervention, our framework still does not report transient fluctuations and slight delays as anomalies.

Based on the evaluation results, we find that our solution is able to correctly identify all the anomalies in all these 5 different cases.

Effectiveness Analysis

In this section, we will delve into the details about the effectiveness and efficiency of our framework. To quantitatively measure the performance, we use F-measure to measure the accuracy and detection time delay to measure the efficiency. The precision and recall in computing F-measure are quantified according to the ground truth shown in Table 5.1. To investigate how λ affects the results, we conducted experiments with various λ values.

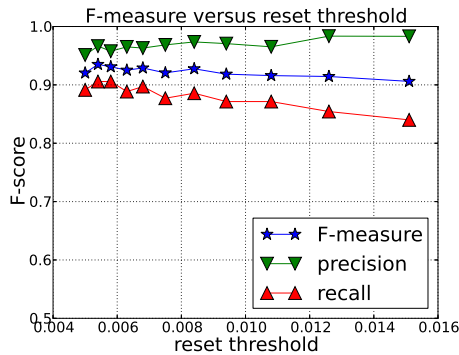


Figure 5.9: F-measure versus reset threshold

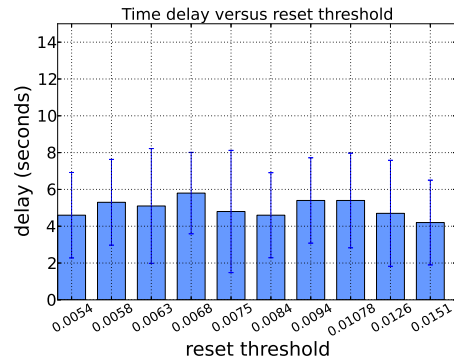


Figure 5.10: Time delay versus reset threshold

The experimental results of how λ affects the results accuracy are illustrated in Figure 5.9. To mitigate the randomness caused by the distributed environment, the precision, recall, and the F-measure are averaged with 10 runs.

As shown, as λ increases, *precision* increases but *recall* decreases. The result shows that the highest F-measure is 0.9351 while the lowest is 0.9060, which is stable. This is due to the changing of precision and recall cancels each other and makes F-measure insensitive to λ .

The reason for the decreasing of recall is as follows: The increase of λ causes the upper bound of stream anomaly scores to decrease and indirectly increases the reset frequency. After each reset of stream anomaly scores, some real anomalies would be skipped and they would reduce the recall. In practice, the low recall does not indicate

that our method misses the real alerts. Figure 5.11 plots the experiment results with the worst recall, where the x-axis denotes the time and the y-axis denotes the number of anomalies. Comparing with the ground truth (the top 6 sub-figures in Figure 5.8), whenever there is an injection, the alerts are generated. Therefore, all the injections can be captured.

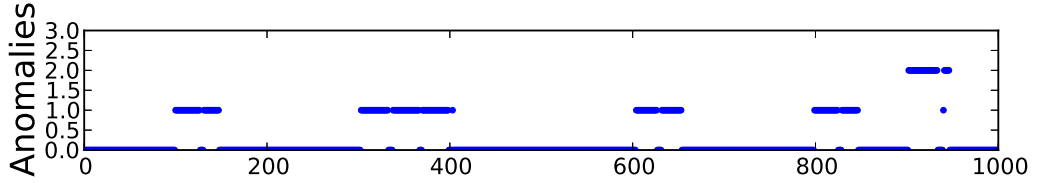


Figure 5.11: Generated alerts with the worst recall

In terms of the time delay, our proposed framework is able to identify the anomalies in real time. As shown in Figure 5.10, the experimental results indicate that the average time delay in all the experiments are less than 6 seconds. We also notice that the variance of the time delay is large, this is because the experiments are conducted in a distributed system, where the environment is highly dynamic. Since the delay consists of network delay, injection execution delay, and the detection delay, the actual delay of our detection method should be less than the observed value.

Results Comparison

To demonstrate the superiority of our framework, we also conduct experiments to identify the anomalies with the same injection settings using the alternative methods including *contextual anomaly detection (CAD)* and *rule-based continuous query (Rule-CQ)*. The contextual anomaly detection is equivalent to the snapshot scoring in our framework. For the rule-based continuous query, we define three rules to capture three types of anomalies, including high CPU utilization (rule 1), low CPU utilization (rule 2), and high memory usage anomalies (rule 3), respectively. Different combinations of the three rules are used in the experiments.

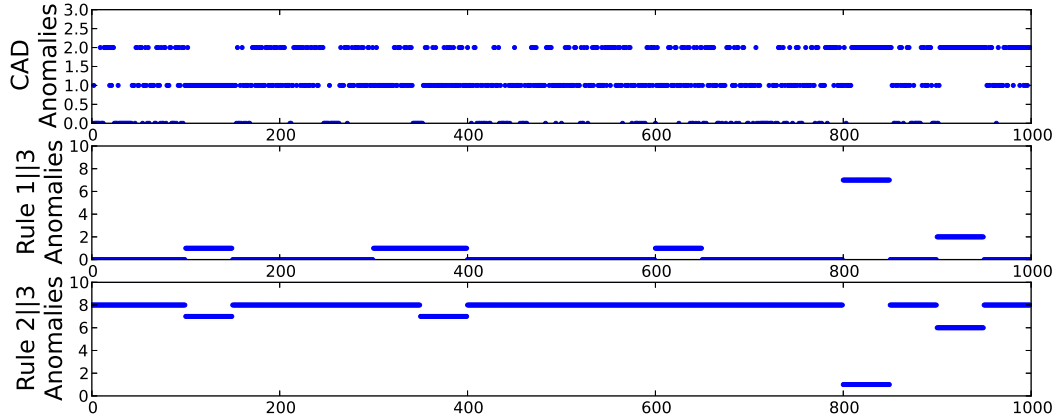


Figure 5.12: Generated alerts by CAD and Rule-CQ

The generated alerts of these methods are shown in Figure 5.12, where the x-axis denotes the time and y-axis denotes the number of anomalies. As illustrated, the contextual anomaly detection method generates a lot of false alerts. This is because this method is sensitive to the transient fluctuation. Once an observation deviates from the others at a timestamp, an alert would be triggered. For Rule-CQ method, we experiment all the combinations and report the results of the two best combinations: C1 (rule 1 or rule 2) and C2 (rule 2 or rule 3). Similarly, the Rule-CQ method also generates many false alerts since it is difficult to use rules to cover all the anomaly situations. Table 5.2 quantitatively shows the precision, recall, and F-measure of the three methods as well as the results of our method. The low-precision and high-recall results of CAD and Rule-CQ indicate that all these method are too sensitive to fluctuations.

Table 5.2: Measures of different methods

Method \ Measure	precision	recall	F-measure
CAD	0.4207	1.0000	0.5922
C1: Rule 1—3	0.5381	1.0000	0.6997
C2: Rule 2—3	0.0469	1.0000	0.0897
Our method (worst case)	0.9832	0.8400	0.9060

A real system problem detected

We have identified a real system problem when deployed our framework on two computing clusters in our department. In one of the clusters, we continuously receive alerts. Logging into the cluster, we find the CPU utilization is high even no tasks are running. We further identify that the high CPU utilization is caused by several processes named *hfsd*. We reported the anomaly to IT support staffs and they confirmed that there exist some problems in this cluster. The high CPU utilization is caused by continuous attempts to connect to a failure node in the network file system. After fixing this problem, these out-of-expectation but real alerts disappear.

5.5.2 Real Word Scenario 2 — Twitter Topics Anomaly Detection

In this section, we conduct experiments on two twitter datasets to perform twitter topic anomaly detection. The first dataset with 7,858,046 tweets was collected by using Twitter Streaming API during 03/09/2011-03/23/2011 and the second dataset with 10,780,000 tweets was collected during 03/23/2012 – 05/28/2012. The first dataset contains the topics of the countries and the second dataset contains topics about candidates of the president election. For pre-processing, the first dataset is aggregated by hour and the second dataset is aggregated by day, and then the change ratios between contiguous timestamps are calculated and feed to our framework as streams. Figure 5.13 and Figure 5.14 illustrate the time series of change ratios as well as the identified anomalies. Also, some of the typical events are marked in both two figures and are described in Table 5.3 and Table 5.4, respectively.

Take the first dataset for example, five sets of anomalies are detected. The anomalies of set 1 is raised by the topic *japan* around March 11th 2011, when a 9 magnitude

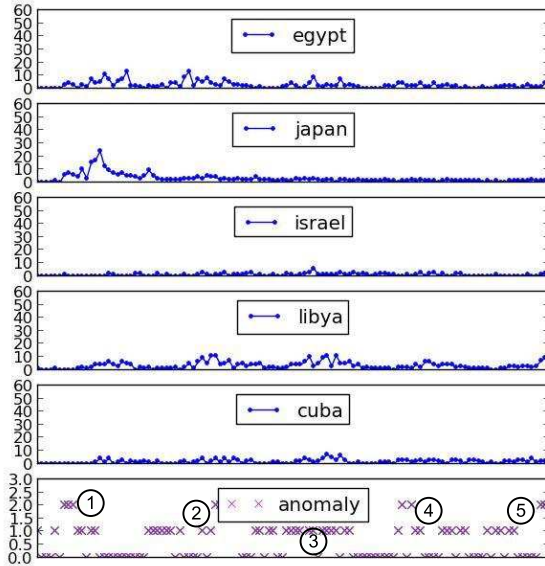


Figure 5.13: Country dataset

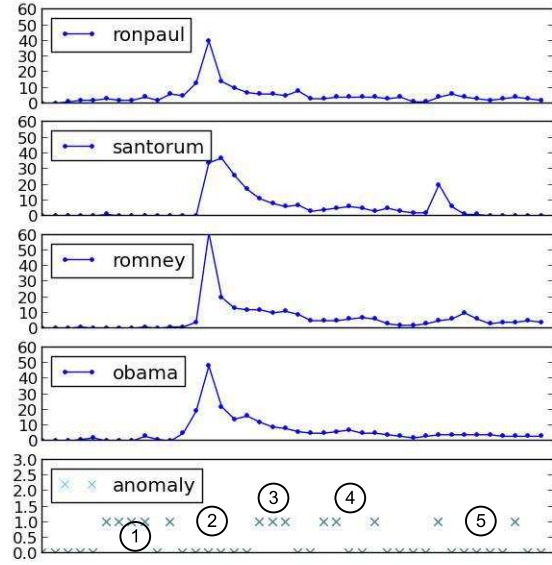


Figure 5.14: Election dataset

Table 5.3: Events in country group

No	Event
1	Japan 9.0 magnitude earthquake. Anti-Gaddafi action.
2	Bloody assaults in Libya. Egypt evolution conflict.
3	Egypt constitutional referendum.
4	Japan is too quiet compare with others.
5	International armed conflict in Libya.

earthquake happened near Japan that day. Almost at the same time, the National Transitional Council flag is flown by anti-Gaddafi fighters in Libya on 10th March 2011. Observing from the figures, the change ratios of these two topics are higher than others during this time period. For this time periods, the collective anomaly detection algorithms generate a large number of false-positives since the change ratios of all the topics are changing irregular all the time. During time period of March 12th-13th, 2011 (anomaly set 2), the change ratios of topic *egypt* and *libya* are high and the anomalies of set 2 is observed. For comparison, The contextual anomaly detection algorithms generate a lot of alerts since fluctuations occur frequently during this time period. According to the news report, several conflicts happened in these two coun-

Table 5.4: Events in election group

No	Event
1	Ron Paul's campaign became active earlier than others.
2	No alert since all topics burst.
3	Santorum's active peak lasts longer than others.
4	False positive alerts.
5	Santorum quited the Republican presidential primaries.

tries due to the revolution. Similarly, other anomaly sets are the follow-up conflicts either happened in Egypt or Libya. All the identified anomalies can be validated by variable information resources.

5.6 Chapter Summary

In this chapter, we propose a real time anomaly detection framework to identify the contextual collective anomalies from a collection of streams. Our proposed method firstly quantifies the snapshot level anomaly of each stream based on the contextual information. Then the contextual information and the historical information are used in combination to quantify the anomaly severity of each stream. Based on the distribution of the stream anomaly scores, an implicit threshold is dynamically calculated and the alerts are triggered accordingly. To demonstrate the usefulness of the proposed framework, several sets of experiments are conducted to demonstrate its effectiveness and efficiency.

CHAPTER 6

Conclusion

Distributed systems are the new trends for people to solve the modern computation problems. As the scale of the systems getting increasingly large, more efforts need to be paid to facilitate people to investigate and manage them. In this dissertation, the problems of leveraging temporal data mining techniques for distributed system management have been discussed. Specifically, three related but orthogonal concrete problems have been studied: 1) The event summarization problem that facilitate the system event analysis; 2) The cloud prediction problem that make the cloud systems more intelligent and enables the autonomous computing; 3) The stream anomaly detection problem that allow the system to self-diagnosis in real time.

For the issue of facilitating system event analysis, the solution of event summarization has been presented. Specifically, a novel event summarization methodology called *NES* which is able to summarize the given event logs with periodical and correlation patterns is proposed. Using *NES*, event analysts are able to obtain a concise yet accurate summary which demonstrates the running status of the system. Besides *NES*, an integrated framework call *META* is proposed. *META* is an event summarization framework that provides various of event operations, include event storage, event multi-resolution analysis and event summarization. It facilitates the event analysts by enabling them end-to-end solutions when analyzing the event logs. Due to its flexibility, the existing and future event summarization methods can be easily plugged into *META*, and therefore making *META* more powerful.

For the issue of making the cloud systems autonomy and more intelligent, we propose the data mining based approach to solve the problems of cloud capacity planning and instant VM provision. Concretely, we abstract and formulate these two problems as the time series prediction problem, then we leverage ensemble time series prediction as well as the VM deprovision probability estimation to predict the demand of VM

provision/deprovision. The experimental evaluation demonstrates the effectiveness and efficacy of the proposed solution.

For the issue of the system self-diagnosis, we propose a real-time streaming anomaly detection algorithm. The proposed algorithm is able to identify a particular anomaly called *contextual-collective* anomaly that occurs frequently in load-balanced distributed systems. Our proposed method conducts the anomaly detection in a 3-step approach, and is able to effectively identify most of the anomalies, including the abnormal stream, the abnormal time range, and the severity, according to the results of experimental evaluation. Most importantly, the proposed method is able to identify the anomalies in real time.

In summary, this dissertation attempts to leverage temporal data mining techniques to resolve the system autonomy and management issues in different aspects. As far as we know, this dissertation is the one of the earliest attempts that solves such issues from the analytic perspective instead from the system perspective.

Based on these initial exploration, we also found several limitation of the proposed works and there are some promising extensions can be done in the future. The current methods for event summarization has limited express power and scalability. For example, it is unable to express the event relationship if there exists event clique. Moreover, it is unable to handle huge event logs as it requires superlinear running time to generate the summarise. In the future, more expressive model and more effective or parallel summarization algorithm would be proposed. For the issue of improving cloud system autonomy, current proposed solutions is incapable of handling sudden demand bursts as the models have limited power to predict such abnormal situation. In the future, advanced burst detection algorithms can be integrated into current prediction framework to boost the solution's stability. In terms of system self-diagnosis, there are also some limitations on the aspects of diagnosis power and scalability. The proposed method is effective to discover the anomalies, but it is

not able to provide an intuitive explanation about the root cause of the anomalies. Moreover, the design of this method is not scalable enough, so it is prohibitive to handle a large number of event streams.

BIBLIOGRAPHY

- [ABB⁺03] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. Stream: the stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 665–665. ACM, 2003.
- [Abd03] Hervé Abdi. Partial least squares regression (pls-regression), 2003.
- [ADGI08] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160. ACM, 2008.
- [AF07] Fabrizio Angiulli and Fabio Fassetti. Detecting distance-based outliers in streams of data. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 811–820. ACM, 2007.
- [AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, pages 207–216. ACM, 1993.
- [Aka69] Hirotugu Akaike. Fitting autoregressive models for prediction. *Annals of the institute of statistical mathematics*, 21(1):243–247, 1969.
- [AKW79] Alfred V Aho, Brian W Kernighan, and Peter J Weinberger. Awka pattern scanning and processing language. *Software: Practice and Experience*, 9(4):267–279, 1979.
- [ALGJ99] Stefan Axelsson, Ulf Lindqvist, Ulf Gustafson, and Erland Jons-son. Approach to unix security logging. *Doktorsavhandlingar vid Chalmers Tekniska Hogskola*, pages 137–158, 1999.
- [AMS⁺96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A Inkeri Verkamo. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996.
- [AP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer, 2002.

- [App] AppFirst. Appfirst monitoring as a service. <http://www.appfirst.com/devops-dashboard/index.php>.
- [AY01] Charu C Aggarwal and Philip S Yu. Outlier detection for high dimensional data. *ACM Sigmod Record*, 30(2):37–46, 2001.
- [BCF⁺05] Douglas Burdick, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Mafia: A maximal frequent itemset algorithm. *Knowledge and Data Engineering, IEEE Transactions on*, 17(11):1490–1504, 2005.
- [BCFL09] Yingyi Bu, Lei Chen, Ada Wai-Chee Fu, and Dawei Liu. Efficient anomaly monitoring over moving object trajectory streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 159–168. ACM, 2009.
- [BCH⁺01] Eric Bloedorn, Alan D Christiansen, William Hill, Clement Skorupka, Lisa M Talbot, and Jonathan Tivel. Data mining for network intrusion detection: How to get started. Technical report, MITRE Technical Report, 2001.
- [BFP⁺73] Manuel Blum, Robert W Floyd, Vaughan Pratt, Ronald L Rivest, and Robert E Tarjan. Time bounds for selection. *Journal of computer and system sciences*, 7(4):448–461, 1973.
- [Bis06] Christopher M Bishop. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.
- [BJR13] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 2013.
- [BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, pages 93–104. ACM, 2000.
- [Bro99] Matt Brown. Event logging system and method for logging events in a network system, January 5 1999. US Patent 5,857,190.
- [Bro04] Robert Goodell Brown. *Smoothing, forecasting and prediction of discrete time series*. Courier Dover Publications, 2004.
- [Car12] David Carasso. Exploring splunk. *CITO Research*. New York, NY, 2012.

- [CBK07] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Outlier detection: A survey. *ACM Computing Surveys*, 2007.
- [CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.
- [CCD⁺03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J Franklin, Joseph M Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R Madden, Fred Reiss, and Mehul A Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM, 2003.
- [CF99] Kin-Pong Chan and AW-C Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE, 1999.
- [CH94] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *arXiv preprint cs/9402102*, 1994.
- [Cha03] Chris Chatfield. *The Analysis of Time Series: An Introduction*. Chapman and Hall, 6 edition, 2003.
- [CMS97] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 558–567. IEEE, 1997.
- [CS02a] Sung-Hyuk Cha and Sargur N Srihari. On measuring the distance between histograms. *Pattern Recognition*, 35(6):1355–1370, 2002.
- [CS02b] Darya Chudova and Padhraic Smyth. Pattern discovery in sequences under a markov assumption. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162. ACM, 2002.
- [CSD98] Soumen Chakrabarti, Sunita Sarawagi, and Byron Dom. Mining surprising patterns using temporal description length. In *VLDB*, volume 98, pages 606–617, 1998.

- [CWO⁺12] Tom Christiansen, Larry Wall, Jon Orwant, et al. *Programming Perl: Unmatched power for text processing and scripting*. O’Reilly Media, Inc., 2012.
- [CYD06] Yuehui Chen, Bo Yang, and Jiwen Dong. Time-series prediction using a local linear wavelet neural network. *Neurocomputing*, 69(4):449–465, 2006.
- [D⁺92] Ingrid Daubechies et al. *Ten lectures on wavelets*, volume 61. SIAM, 1992.
- [Dav07] Salomon David. *Data Compression: The Complete Reference*. Springer, 2007.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [DGHS95] Paul Dagum, Adam Galper, Eric Horvitz, and Adam Seiver. Uncertain reasoning and forecasting. *International Journal of Forecasting*, 11(1):73–87, 1995.
- [DJCR01] Mc C Deo, A Jha, AS Chaphekar, and K Ravikant. Neural networks for wave forecasting. *Ocean Engineering*, 28(7):889–898, 2001.
- [DP07] Guozhu Dong and Jian Pei. Mining partial orders from sequences. In *Sequence Data Mining*, pages 89–112. Springer, 2007.
- [Eng82] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.
- [Eve14] EventId. <http://www.eventid.net>, 2014.
- [FC98] Julian Faraway and Chris Chatfield. Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(2):231–250, 1998.
- [Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

- [FY⁺49] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, 1949.
- [GC03] Alan G Ganek and Thomas A Corbi. The dawning of the autonomic computing era. *IBM systems Journal*, 42(1):5–18, 2003.
- [GGW10] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.
- [GHM⁺09] Albert Greenberg, James Hamilton, David Maltz, , and Parveen Patel. The cost of a cloud: Research problems in data center networks. In *Computer Communication Review*, 2009.
- [GHP⁺03] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S Yu. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.
- [GRGM09] Zhenhuan Gong, Prakash Ramaswamy, Xiaohui Gu, and Xiaosong Ma. Siglm: Signature-driven load management for cloud computing infrastructures. In *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pages 1–9. IEEE, 2009.
- [Grü07] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.
- [GTDVMFV09] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.
- [GXZ⁺10] Yong Ge, Hui Xiong, Zhi-hua Zhou, Hasan Ozdemir, Jannite Yu, and Kuo Chu Lee. Top-eye: Top-k evolving trajectory outlier detection. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1733–1736. ACM, 2010.
- [GZ01] Karam Gouda and M Zaki. Efficiently mining maximal frequent itemsets. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 163–170. IEEE, 2001.

- [HA04] Victoria J Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.
- [Haa10] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.
- [Ham09] James Hamilton. Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services. In *CIDR*, 2009.
- [Har90] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.
- [HC08] Kuo-Yu Huang and Chia-Hui Chang. Efficient mining of frequent episodes from complex sequences. *Information Systems*, 33(1):96–114, 2008.
- [HCXY07] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
- [HHWB02] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.
- [HKP06] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.
- [Hor01] Paul Horn. Autonomic computing: Ibm’s perspective on the state of information technology. *Technic Report*, 2001.
- [HPMA⁺00] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM, 2000.
- [HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, pages 1–12. ACM, 2000.

- [HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.
- [HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*, volume 2. Springer, 2009.
- [Ing93] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and computer modelling*, 18(11):29–57, 1993.
- [Ing03] Ching-Kang Ing. Multistep prediction in autoregressive processes. *Econometric Theory*, 19(02):254–279, 2003.
- [IV06] Renáta Iváncsy and István Vajk. Frequent pattern mining in web log data. *Acta Polytechnica Hungarica*, 3(1):77–90, 2006.
- [JD02] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375. ACM, 2002.
- [JHP⁺09] Weihang Jiang, Chongfeng Hu, Shankar Pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. Understanding customer problem troubleshooting from storage system logs. In *FAST*, volume 9, pages 43–56, 2009.
- [Joh89] David B Johnson. Distributed system fault tolerance using message logging and checkpointing. Technical report, DTIC Document, 1989.
- [JPL11] Yexi Jiang, Chang-Shing Perng, and Tao Li. Natural event summarization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 765–774. ACM, 2011.
- [JPLC11] Yexi Jiang, Chang-shing Perng, Tao Li, and Rong Chang. Asap: A self-adaptive prediction system for instant cloud resource demand provisioning. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1104–1109. IEEE, 2011.
- [JPLC12a] Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong Chang. Intelligent cloud capacity management. In *Network Operations and Man-*

- agement Symposium (NOMS), 2012 IEEE, pages 502–505. IEEE, 2012.
- [JPLC12b] Yexi Jiang, Chang-shing Perng, Tao Li, and Rong Chang. Self-adaptive cloud capacity planning. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 73–80. IEEE, 2012.
- [JU97] Simon J Julier and Jeffrey K Uhlmann. A new extension of the kalman filter to nonlinear systems. *Int. symp. aerospace/defense sensing, simul. and controls*, 3(26):3–2, 1997.
- [KB96] Ieabeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, 1996.
- [KC03] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
- [KH00] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with applications*, 19(2):125–132, 2000.
- [KL99] Tom M Kroeger and Darrell DE Long. The case for efficient file access pattern modeling. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 14–19. IEEE, 1999.
- [KL01] Tom M Kroeger and Darrell DE Long. Design and implementation of a predictive file prefetching algorithm. In *USENIX Annual Technical Conference, General Track*, pages 105–118, 2001.
- [KR94] Thomas Kolarik and Gottfried Rudorfer. Time series forecasting using neural networks. *ACM Sigapl Apl Quote Quad*, 25(1):86–94, 1994.
- [KT08] Jerry Kiernan and Evimaria Terzi. Constructing comprehensive summaries of large event sequences. In *KDD*. ACM, 2008.
- [KT09] Jerry Kiernan and Evimaria Terzi. Constructing comprehensive summaries of large event sequences. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):21, 2009.

- [LB00] Jianxiong Luo and Susan M Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(8):687–703, 2000.
- [LBV01] Jianxiong Luo, Susan Bridges, and RB Vaughn. Fuzzy frequent episodes for real-time intrusion detection. In *FUZZ-IEEE*, pages 368–371, 2001.
- [Lew00] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual Meeting of the Society for Academic Emergency Medicine in San Francisco, California*, pages 1–14. Citeseer, 2000.
- [LGS⁺08] Petroula Louka, Georges Galanis, Nils Siebert, Georges Kariniotakis, Petros Katsafados, I Pytharoulis, and G Kallos. Improvements in wind speed forecasts for wind power prediction purposes using kalman filtering. *Journal of Wind Engineering and Industrial Aerodynamics*, 96(12):2348–2362, 2008.
- [LHL08] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 140–149. IEEE, 2008.
- [LK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.
- [LKL07] David Lo, Siau-Cheng Khoo, and Chao Liu. Efficient mining of iterative patterns for software specification discovery. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 460–469. ACM, 2007.
- [LLMP05] Tao Li, Feng Liang, Sheng Ma, and Wei Peng. An integrated framework on mining logs files for computing system management. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 776–781. ACM, 2005.
- [LLZO02] Tao Li, Qi Li, Shenghuo Zhu, and Mitsunori Ogihara. A survey on wavelet applications in data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):49–68, 2002.

- [LMT⁺04] Francois Labonte, Peter Mattson, William Thies, Ian Buck, Christos Kozyrakis, and Mark Horowitz. The stream virtual machine. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 267–277. IEEE Computer Society, 2004.
- [Log] Loggly. Loggly log management tool. <https://www.loggly.com/product/>.
- [LOP06] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Fast and memory efficient mining of frequent closed itemsets. *Knowledge and Data Engineering, IEEE Transactions on*, 18(1):21–36, 2006.
- [LPP⁺10] Tao Li, Wei Peng, Charles Perng, Sheng Ma, and Haixun Wang. An integrated data-driven framework for computing system management. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40(1):90–99, 2010.
- [LR85] Richard Lewis and Gregory C Reinsel. Prediction of multivariate time series by autoregressive model fitting. *Journal of multivariate analysis*, 16(3):393–411, 1985.
- [LSU05] Srivatsan Laxman, PS Sastry, and KP Unnikrishnan. Discovering frequent episodes and learning hidden markov models: A formal connection. *Knowledge and Data Engineering, IEEE Transactions on*, 17(11):1505–1517, 2005.
- [LX01] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 130–143. IEEE, 2001.
- [Mal89] Stephen Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE PAMI*, 11, 1989.
- [ME10] Nizar R Mabroukeh and Christie I Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR)*, 43(1):3, 2010.
- [MH01] Sheng Ma and Joseph L Hellerstein. Mining partially periodic event patterns with unknown periods. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 205–214. IEEE, 2001.

- [Mit97] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.
- [MLH10] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 41–48. IEEE, 2010.
- [MLS10] Shicong Meng, Ling Liu, and Vijayaraghavan Soundararajan. Tide: achieving self-scaling in virtualized datacenter management middleware. In *Proceedings of the 11th International Middleware Conference Industrial track*, pages 17–22. ACM, 2010.
- [MR04] Nicolas Méger and Christophe Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *Knowledge Discovery in Databases: PKDD 2004*, pages 313–324. Springer, 2004.
- [MS01] Heikki Mannila and Marko Salmenkivi. Finding simple intensity descriptions from event sequence data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 341–346. ACM, 2001.
- [MSB10] Swarna Mylavarapu, Vijay Sukthankar, and Pradipta Banerjee. An optimized capacity planning approach for virtual infrastructure exhibiting stochastic workload. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 386–390. ACM, 2010.
- [MTDB04] Marco Martinelli, Enrico Tronci, Giovanni Dipoppa, and Claudio Balducelli. Electric power system anomaly detection using neural networks. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 1242–1248. Springer, 2004.
- [MTT08] Stephen Marsland, Carole J Twining, and Chris J Taylor. A minimum description length objective function for groupwise non-rigid image registration. *Image and Vision Computing*, 26(3):333–346, 2008.
- [MWBF98] Nancy E Miller, Pak Chung Wong, Mary Brewster, and Harlan Foote. Topic islands tm-a wavelet-based text visualization system. In *Visualization’98. Proceedings*, pages 189–196. IEEE, 1998.
- [MZZ12] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. High-performance complex event processing over xml streams. In *Proceedings of the*

2012 ACM SIGMOD International Conference on Management of Data, pages 253–264. ACM, 2012.

- [OS07] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 575–584. IEEE, 2007.
- [Ped97] Elazar J Pedhazur. *Multiple regression in behavioral research: Explanation and prediction*. Holt, Rinehart and Winston, 1997.
- [PG07] Jatin D Parmar and Sanjay Garg. Modified web access pattern (mwap) approach for sequential pattern mining. *INFOCOMP Journal of Computer Science*, 6(2):46–54, 2007.
- [PH05] Ping-Feng Pai and Wei-Chiang Hong. Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Conversion and Management*, 46(17):2669–2688, 2005.
- [PH06] Ping-Feng Pai and Wei-Chiang Hong. Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software*, 79(6):747–755, 2006.
- [PHM⁺00] Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, pages 21–30, 2000.
- [PHMAZ00] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *Knowledge Discovery and Data Mining. Current Issues and New Applications*, pages 396–407. Springer, 2000.
- [PHW02] Jian Pei, Jiawei Han, and Wei Wang. Mining sequential patterns with constraints in large databases. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 18–25. ACM, 2002.
- [PIL06] Anantha M Prasad, Louis R Iverson, and Andy Liaw. Newer classification and regression tree techniques: bagging and random forests for ecological prediction. *Ecosystems*, 9(2):181–199, 2006.

- [PKG03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 315–326. IEEE, 2003.
- [PLL07] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. Incremental local outlier detection for data streams. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 504–515. IEEE, 2007.
- [PLM05] Wei Peng, Tao Li, and Sheng Ma. Mining logs files for data-driven system management. *ACM SIGKDD Explorations Newsletter*, 7(1):44–51, 2005.
- [PM02] Ivan Popivanov and Renee J Miller. Similarity search over time-series data using wavelets. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 212–221. IEEE, 2002.
- [PMSR09] Debprakash Patnaik, Manish Marwah, Ratnesh Sharma, and Naren Ramakrishnan. Sustainable operation and management of data center chillers using temporal data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1305–1314. ACM, 2009.
- [PP88] Peter CB Phillips and Pierre Perron. Testing for a unit root in time series regression. *Biometrika*, 75(2):335–346, 1988.
- [PPC+01] Jian Pei, Helen Pinto, Qiming Chen, Jiawei Han, Behzad Mortazavi-Asl, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.
- [PPLW07] Wei Peng, Charles Perng, Tao Li, and Haixun Wang. Event summarization for system management. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1028–1032. ACM, 2007.
- [PR81] Robert S Pindyck and Daniel L Rubinfeld. *Econometric models and economic forecasts*, volume 2. McGraw-Hill New York, 1981.

- [Pre03] James E Prewett. Analyzing cluster log files using logsurfer. In *Proceedings of the 4th Annual Conference on Linux clusters*. Citeseer, 2003.
- [PTG⁺03] Chang-Shing Perng, David Thoenen, Genady Grabarnik, Sheng Ma, and Joseph Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 729–734. ACM, 2003.
- [RT74] Dennis M Ritchie and Ken Thompson. The unix time-sharing system. *Communications of the ACM*, 17(7):365–375, 1974.
- [Sch98] Thomas Schreiber. Constrained randomization of time series data. *Physical Review Letters*, 80(10):2105, 1998.
- [SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, volume 98, pages 428–439, 1998.
- [SG08] Mahboobeh Soleimani and Ali A Ghorbani. Critical episode mining in intrusion detection alerts. In *Communication Networks and Services Research Conference, 2008. CNSR 2008. 6th Annual*, pages 157–164. IEEE, 2008.
- [Spl] Splunk. <http://www.splunk.com/>.
- [SS09] Nicholas I Sapankevych and Ravi Sankar. Time series prediction using support vector machines: a survey. *Computational Intelligence Magazine, IEEE*, 4(2):24–38, 2009.
- [TC12] Nikolaj Tatti and Boris Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.
- [TdAF91] Zaiyong Tang, Chrys de Almeida, and Paul A Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *Simulation*, 57(5):303–310, 1991.
- [TRH01] David Thoenen, Jim Riosa, and Joseph L Hellerstein. Event relationship networks: a framework for action oriented analysis in event management. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 593–606. IEEE, 2001.

- [TTD⁺08] Liang Tang, Chang-jie Tang, Lei Duan, Chuan Li, Ye-xi Jiang, Chun-qiu Zeng, and Jun Zhu. Movstream: An efficient algorithm for monitoring clusters evolving in data streams. In *Granular Computing, 2008. GrC 2008. IEEE International Conference on*, pages 582–587. IEEE, 2008.
- [TTJ⁺08] Liang Tang, Changjie Tang, Yexi Jiang, Chuan Li, Lei Duan, Chun-qiu Zeng, and Kaikuo Xu. Troadgrid: An efficient trajectory outlier detection algorithm with grid-based space division. In *NDBC*, 2008.
- [TY06] Jun-ichi Takeuchi and Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *Knowledge and Data Engineering, IEEE Transactions on*, 18(4):482–492, 2006.
- [Vaa04] Risto Vaarandi. A breadth-first algorithm for mining frequent patterns from event logs. In *Intelligence in Communication Systems*, pages 293–308. Springer, 2004.
- [Val90] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.
- [VGSB⁺01] Tony Van Gestel, Johan AK Suykens, D-E Baestaens, Annemie Lambrechts, Gert Lanckriet, Bruno Vandaele, Bart De Moor, and Joos Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *Neural Networks, IEEE Transactions on*, 12(4):809–821, 2001.
- [Vie] Windows Event Viewer. <http://windows.microsoft.com/en-us/windows/open-event-viewer#1TC=windows-7>.
- [WF05] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [WH04] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 79–90. IEEE, 2004.
- [WK90] Richard S Wallace and Takeo Kanade. Finding natural clusters having minimum description length. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume 1, pages 438–442. IEEE, 1990.

- [WWLW10] Peng Wang, Haixun Wang, Majin Liu, and Wei Wang. An algorithmic approach to event summarization. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 183–194. ACM, 2010.
- [XHF⁺08] Wei Xu, Ling Huang, Armando Fox, David A Patterson, and Michael I Jordan. Mining console logs for large-scale system problem detection. *SysML*, 8:4–4, 2008.
- [XHF⁺09] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. ACM, 2009.
- [XHF⁺10] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Experience mining googles production console logs. In *Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, 2010.
- [XTL⁺10] Kaikuo Xu, Changjie Tang, Chuan Li, Yexi Jiang, and Rong Tang. An mdl approach to efficiently discover communities in bipartite network. In *Database Systems for Advanced Applications*, pages 595–611. Springer, 2010.
- [Xu10] Wei Xu. System problem detection by mining console logs. *PhD Thesis, UC Berkeley*, 2010.
- [YH02] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.
- [YLJW10] Yanfang Ye, Tao Li, Qingshan Jiang, and Youyu Wang. Cimds: adapting postprocessing techniques of associative classification for malware detection. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(3):298–307, 2010.
- [YWL⁺08] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang. An intelligent pe-malware detection system based on association mining. *Journal in Computer Virology*, 4(4):323–334, 2008.
- [YWLY07] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. Imds: Intelligent malware detection system. In *Proceedings of the 13th ACM*

SIGKDD international conference on Knowledge discovery and data mining, pages 1043–1047. ACM, 2007.

- [ZB03] Qiankun Zhao and Sourav S Bhowmick. Sequential pattern mining: A survey. *Technical Report CAIS Nayang Technological University Singapore*, pages 1–26, 2003.
- [ZH05] Mohammed Javeed Zaki and C-J Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *Knowledge and Data Engineering, IEEE Transactions on*, 17(4):462–478, 2005.
- [Zha04] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.
- [ZHF06] Baoyao Zhou, Siu Cheung Hui, and Alvis Cheuk Ming Fong. Efficient sequential access pattern mining for web recommendations. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 10(2):155–168, 2006.
- [ZJZ⁺13] Chunqiu Zeng, Yexi Jiang, Li Zheng, Jingxuan Li, Lei Li, Hongtai Li, Chao Shen, Wubai Zhou, Tao Li, Bing Duan, et al. Fiu-miner: a fast, integrated, and user-friendly system for data mining in distributed environment. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1506–1509. ACM, 2013.
- [ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 1978.
- [ZL01] Osmar R Zaiane and J Luo. Web usage mining for a better web-based learning environment. In *Proceedings of conference on advanced technology for education*, pages 60–64. Citeseer, 2001.
- [ZLC10] Wenzhi Zhou, Hongyan Liu, and Hong Cheng. Mining closed episodes from event sequences efficiently. In *Advances in Knowledge Discovery and Data Mining*, pages 310–318. Springer, 2010.
- [ZS02] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th*

international conference on Very Large Data Bases, pages 358–369. VLDB Endowment, 2002.

- [ZTL⁺04] Jie Zuo, Chang-jie Tang, Chuan Li, Chang-an Yuan, and An-long Chen. Time series prediction based on gene expression programming. In *Advances in web-age information management*, pages 55–64. Springer, 2004.

VITA

YEXI JIANG

2010 – Now	Ph.D., Computer Science Florida International University Miami, Florida, U.S.A.
2010	M.S. and B.E., Computer Science Sichuan University Chengdu, Sichuan, China

PUBLICATIONS

1. Yexi Jiang and Tao Li. Data mining application in cloud computing. In Data Mining Where Theory Meets Practice. Xiamen University Press, 2013, Page 38-64, ISBN 9787561542941.
2. Yexi Jiang, Chunqiu Zeng, Jian Xu, Tao Li. “Real time contextual collective anomaly detection over multiple data streams”. ACM SIGKDD Workshop on Outlier Detection & Description under Data Diversity (SIGKDD Workshop ODD²), 2014.
3. Liang Tang, Yexi Jiang, Lei Li, Tao Li. “Ensemble Contextual Bandits for Personalized Recommendation”, ACM Conference on Recommender Systems (RecSys), Pages 73 - 80, 2014.
4. Lei Li, Chao Shen, Long Wang, Li Zheng, Yexi Jiang, Liang Tang, Hongtai Li, Longhui Zhang and Chunqiu Zeng. “iMiner: Mining Inventory Data for Intelligent Management”. ACM Conference on Information and Knowledge Management (CIKM), Pages 2057 - 2059, 2014.
5. Yexi Jiang, Chang-shing Perng, Tao Li. “META: Multi-resolution Framework for Event Summarization”, SDM International Conference on Data Mining (SDM), Pages 605 - 613, 2014.
6. Li Zheng, Chunqiu Zeng, Lei Li, Yexi Jiang, Wei Xue, Jingxuan Li, Chao Shen, Wubai Zhou, Hongtai Li, Liang Tang, Tao Li, Bing Duan, Ming Lei, Pengnian Wang. “Applying Data Mining Techniques to Address Critical Process Optimization Needs in Advanced Manufacturing”, ACM Conference on Knowledge Discovery and Data Mining (SIGKDD), Pages 1739 - 1748, 2014.
7. Yexi Jiang, Chang-shing Perng, Tao Li, Rong Chang. “Cloud Analytics for Capacity Planning and Instant VM Provisioning”, IEEE Transactions on Network and Service Management (TNSM), Volume 10, Issue 3, Pages 312 - 325, 2013.
8. Liang Tang, Tao Li, Yexi Jiang, Zhiyuan Chen. “Dynamic Query Forms for Database Queries”, IEEE Transactions on Knowledge and Data Engineering (TKDE), Volume 26, Issue 9, Pages 2166 - 2178, 2013.

9. Chunqiu Zeng, Yexi Jiang, Li Zhen, Jingxuan Li, Lei Li, Hongtai Li, Chao Shen, Wubai Zhou, Tao Li, Bing Duan, Ming Lei, Pengnian Wang. "FIU-Miner: A Fast, Integrated, and User-Friendly System for Data Mining in Distributed Environment". ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD), Page 1506 - 1509, 2013.
10. Yexi Jiang, Chang-shing Perng, Tao Li, Rong Chang. "Self-adaptive Cloud Capacity Planning". International Conference on Service Computing (SCC), Pages 73 - 80, 2012.
11. Yexi Jiang, Chang-shing Perng, Tao Li, Rong Chang. "Intelligent Cloud Capacity Management". IEEE Network Operations and Management Symposium (NOMS), Pages 502 - 505, 2012.
12. Yexi Jiang, Chang-shing Perng, Tao Li, Rong Chang. "ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning". IEEE International Conference on Data Mining (ICDM), Pages 1104 - 1109, 2011.
13. Yexi Jiang, Chang-shing Perng, Tao Li. "Natural Event Summarization". ACM Conference on Information and Knowledge Management (CIKM), Pages 765 - 774, 2011.
14. Mingjie Tang, Weihang Wang, Yexi Jiang, Yuanchun Zhou, Jinyan Li, Peng Cui, Ying Liu and Baoping Yan. "Bird Bring Flues? Mining Frequent and High Weighted Cliques from Birds Migration Network". Database System for Advanced Application (DASFAA), Pages 359 - 369. 2010.
15. Kaikuo Xu, Changjie Tang, Chuan Li, Yexi Jiang, Rong Tang. "An MDL Approach to Efficiently Discover Communities in Bipartite Network". Database System for Advanced Application (DASFAA), Pages 595 - 611, 2010.
16. Yexi Jiang, Changjie Tang, Kaikuo Xu, Yu Chen, Jie Gong, Liang Tang. "CTSC: Core-Tag oriented Spectral Clustering Algorithm on Web2.0 Tags". The International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Pages 460 - 464, 2009.
17. Yexi Jiang, Changjie Tang, Kaikuo Xu, Lei Duan, Liang Tang, Jie Gong, Chuan Li. "Core-Tag Clustering for Web 2.0 based on Multi-Similarity Measurements". The International Workshop on DataBase and Information Retrieval & Aspects in Evaluating Holistic Quality of Ontology-based Information Retrieval (ApWeb Workshop), Pages 222 - 233, 2009.
18. Liang Tang, Changjie Tang, Lei Duan, Yexi Jiang. "SLICE: A Novel Method to Find Local Linear Correlations by Constructing Hyperplanes". The Joint International Conferences on Asia-Pacific Web Conference and Web-Age Information Management (WAIM), Pages 635 - 640, 2009.
19. KaiKuo Xu, Chen Yu, Yexi Jiang, Rong Tang, Jie Gong, Chuan Li. "A Comparative Study of Correlation Measurements for Searching Similar Tags". International Conference on Advanced Data Mining and Applications (ADMA), Pages 709 - 716, 2008.