

IMPROVING CLOUD SYSTEM RELIABILITY USING
AUTONOMOUS AGENT TECHNOLOGY

Yuanyao Liu

Under the Supervision of Dr. Ausif Mahmood and Dr. Zhengping Wu

DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

AND ENGINEERING

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT

CONNECTICUT

Dec, 2015

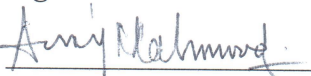

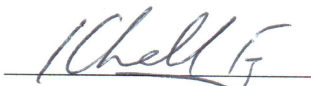
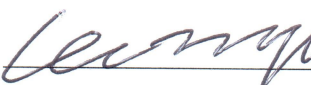
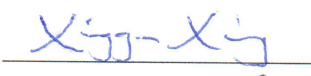

IMPROVING CLOUD SYSTEM RELIABILITY USING AUTONOMOUS AGENT TECHNOLOGY

Yuanyao Liu

Under the Supervision of Dr. Zhengping Wu and Dr. Ausif Mahmood

Approvals

Committee Members

Name	Signature	Date
Dr. Ausif Mahmood		11-19-2015
Dr. Zhengping Wu		10/5/2015
Dr. Khaled M. Elleithy		11/19/2015
Dr. Jeongkyu Lee		11/19/2015
Dr. Xinguo Xiong		11/19/2015
Dr. Qingquan Sun		10-06-2015

Ph.D. Program Coordinator

Dr. Khaled M. Elleithy		11/24/2015
------------------------	--	------------

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood		11-19-2015
-------------------	--	------------

Dean, School of Engineering

Dr. Tarek M. Sobh		12-9-2015
-------------------	--	-----------

IMPROVING CLOUD SYSTEM RELIABILITY USING
AUTONOMOUS AGENT TECHNOLOGY

© Copyright by Yuanyao Liu 2015

IMPROVING CLOUD SYSTEM RELIABILITY USING AUTONOMOUS AGENT TECHNOLOGY

ABSTRACT

Cloud computing platforms provide efficient and flexible ways to offer services and computation facilities to users. Service providers acquire resources according to their requirements and deploy their services in cloud. Service consumers can access services over networks. In cloud computing, virtualization techniques allow cloud providers provide computation and storage resources according to users' requirement. However, reliability in the cloud is an important factor to measure the performance of a virtualized cloud computing platform. Reliability in cloud computing includes the usability and availability. Usability is defined as cloud computing platform provides functional and easy-to-use computation resources to users. In order to ensure usability, configurations and management policies have to be maintained and deployed by cloud computing providers. Availability of cloud is defined as cloud computing platform provides stable and reliable computation resources to users. My research concentrates on improving usability and availability of cloud computing platforms. I proposed a customized agent-based reliability monitoring framework to increase reliability of cloud computing.

ACKNOWLEDGEMENTS

My thanks are wholly devoted to God who has helped me all the way to complete this work successfully. I owe a debt of gratitude to my family for understanding and encouragement.

Then, I would like to express my deepest appreciation to my advisor Dr. Ausif Mahmood and Dr. Zhengping Wu for leading me and guiding me throughout this entire path. They give me the opportunity to be creative. They encouraged me at difficult times and offered valuable suggestions when I faced tough challenges. This dissertation would not have been possible without their supervision. Second, to my committee members, Dr. Khaled M. Elleithy, Dr. Jeongkyu Lee, Dr. Xinguo Xiong and Dr. Sun. Thank you for your encouraging and constructive feedback. Reviewing a thesis is never an easy task, and I am grateful for their valuable and insightful comments.

Third, to the faculty and staff of School of Engineering at University of Bridgeport. I am proud to be a member of this family. Thank you for helping me to develop the skills I need to complete this thesis. Finally yet importantly, this dissertation would not have been possible without the love and support of my family. Thank you all for your encouragement.

TABLE OF CONTENTS

ABSTRACT.....	iv
ACKNOWLEDGEMENTS.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: LITERATURE SURVEY.....	6
2.1 Reliability in the Cloud.....	6
2.2 Temporal Logic and Policy Analysis.....	15
2.3 Event Learning and Prediction.....	18
CHAPTER 3: RESEARCH PLAN OVERVIEW.....	21
3.1 An Autonomous Agent Architecture for Reliability.....	21
CHAPTER 4: SERVICE POLICY ANALYSIS.....	24
4.1 Policy Modeling.....	24
4.1.1 General Policy Model.....	24
4.1.2 Policy Representation Using Temporal Logic.....	29
4.2 Knowledge-Augmented Temporal Logic.....	35
4.2.1 Semantic Extension.....	35
4.2.2 Relationship and Entity.....	41
4.3 Rules in Conflict Analysis and Conflict Reconciliation:.....	50
4.4 Experiment of Policy Conflict Analysis Component.....	53
CHAPTER 5 NON-INTRUSIVE LOG PROCESSING.....	57
5.1 Service Event Pattern Learning.....	57
5.1.1 Event Pattern Learning Technique Survey.....	57

5.1.2 Hash Table with Reversed Frequent Pattern Tree	60
5.2 Event Pattern Detection	68
5.3 Event Pattern Prediction Algorithm.....	70
5.3.1 Filtered-Multi Dimensions Neural Network	70
5.4 Experiments	75
5.5 Summary	79
CHAPTER 6: CONCLUSION	81
6.1 Conclusion	81
6.2 Future Work	82
REFERENCES	83

LIST OF TABLES

Table 4.1	Comparison of Different Policy Conflict Analysis Algorithms (A)	54
Table 4.2	Comparison of Different Policy Conflict Analysis Algorithms (B)	56

LIST OF FIGURES

Figure 3.1	Autonomous Agent Architecture for Reliability Monitoring	23
Figure 4.1	Service in Different Domain	25
Figure 5.1	Structure of a Node	60
Figure 5.2	Reversed Frequent Pattern Tree	61
Figure 5.3	Reversed Pattern Tree using linked list node	62
Figure 5.4	Example of Hash Table and Linked List	63
Figure 5.5	Reversed Pattern Tree with Child Pointers and Sibling Pointers	64
Figure 5.6	New Frequent Pattern Insertion Algorithm	65
Figure 5.7	Reversed Frequent Pattern Tree Compress Algorithm	67
Figure 5.8	Event Sequence Recognition Example	69
Figure 5.9	A Simple Wavelet Neural Network	71
Figure 5.10	Layers of Filtered MD Wavelet Neural Network	72
Figure 5.11	Prediction Accuracy Comparison of WNN and AODE	77
Figure 5.12	The performance of WNN, Multi-Dimension WNN (MD-WNN) and Filter-Multi-Dimension WNN (Filter MD-WNN) in RMSE for training data set with different hidden nodes	78
Figure 5.13	The average precision and recall of different prediction algorithms	78
Figure 5.14	The average time cost of different prediction algorithms	79

CHAPTER 1: INTRODUCTION

1.1 Research Problem and Scope

Cloud computing provides an infrastructure to support efficient and flexible computing resource for service providers and service consumers. Cloud is constructed on various computer systems over communication networks, which provides computing resources through a unified computing platform. Different types of services are provided through this unified platform, such as Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), etc. These services fulfill service consumers' requirements through unified network interfaces. Service consumers choose services based on various needs. Sometimes, a single service can provide enough functionality for different service consumers. Sometimes consumers need several services to finish their tasks. In this situation, consumers have to integrate these services, and service providers need to coordinate with each other. In service coordination, service consumers face different issues and problems. These problems are obstacles that stand in the way between services and consumers. Therefore, service providers intend to provide services that have been integrated to meet consumers' needs. This makes the nature of cloud computing, which provides on-demand and measured services. In a virtualized environment, services are usually deployed in virtual machines or shared virtual machines. Virtualization is one of the most important technologies in cloud computing, which is used to increase the

utilization of physical hardware resources and network bandwidth. Web application providers have the potential to scale virtual resources up and down to achieve cost-effective outcomes. In such virtualized environment, users cannot tell the difference between virtualized computer hardware and physical hardware. Users expect the same capability from real physical hardware. Therefore, virtualized systems receive all types of requests and commands. These requests and commands are finally passed to the host operating system or hypervisor to allocate system resources and perform the computation. Host operating systems or hypervisors are foundation of the upper level virtual machines. Events happened in virtual machines will affect host operating systems or hypervisors. These events may also affect each other in certain ways.

Regardless of functions and scales of services, service consumers need reliable and stable services through service providers. The system reliability is one of the most important requirements for cloud service providers. Reliability means proper functioning of the system under the full range of conditions experienced in the field [1]. However, the definition of reliable service may be different according to different needs. Service consumers consider reliability as proper functioning, security, and ease of use. Service providers also consider reliability in service creation, deployment, integration and separation. The reliability in cloud computing environment also includes providing proper functioning in different stages in service lifecycle. Service integration and separation allow service providers to offer both full set of functionality and part of functionality to service consumers according to service level agreements. The reliability covers various aspects of cloud computing. The base line of the reliability is to provide functioning services.

However, the cloud providers cannot guarantee 100% reliability. For example, Amazon's data center in northern Virginia was down because of power outage [2]. This incident is caused by a weather condition. There is another service outage happened in October 2012, which is caused by a hardware failure. The failure event happened in a single server, and was propagated to the entire system through a chain of software and hardware events. If there is a mechanism, which could monitor servers to prevent this kind of situation, the service providers should be able to provide more reliable services.

Individual services and integrated services face different issues in cloud computing environment. The scalability of services in integrated services is different from single services. For example, communication and dependency may need extra resources for integrated services. In order to ensure the reliability of integrated services, every component in an integrated service not only has to perform proper function but also can cooperate with each other. In order to ensure cooperation among different services, configuration and management policies from different services must be analyzed and enforced according to service level agreements. Policy analysis is a must during this service integration process.

Each service may reside or be deployed in one or more virtual machines. The status of virtual machines reflects the status of running services. The reliability of virtual machines is an essential component for service reliability. When virtual machines are down for any reason, services running on these virtual machines are down. A case worth learning is Heroku [3]. Heroku is a PaaS for Ruby programming. At the time of service outage, there are about 44,000 running applications in its services. Heroku runs its service on

Amazon's EC2 instances. This lesson taught Heroku that separating its services into different availability zone is so crucial for its reliability.

In a virtualized environment, virtual machines created in one physical machine share resources in this physical machine under service level agreements. Virtual machines acquire their required resources from the shared physical machine. Although every virtual machine is independent for users, the entire physical environment is integrated. When users utilize virtual machines or services, resources are usually allocated through virtual machine monitors or hypervisors. Virtual machine monitors do not only allocate resources but also keep track of execution status for virtual machines. System events in virtual machines can be system calls, errors, software failures or system critical failures. Errors and failures are critical events. System critical events may affect the health of virtual machines. Sometimes, system critical events may also affect the reliability of the entire virtualized environment. When one virtual machine crashes, physical machine needs computing and storage resources to restore the crashed one. Restore processes do need extra resources other than running a stable virtual machine. In order to increase the reliability of virtual machines, there must be very effective mechanisms in cloud systems to either avoid system failures and faults or adjust systems so that more serious failures and faults can be prevented.

1.2 Potential Contributions of the Proposed Research

I developed an autonomous agent architecture using knowledge-augmented temporal logic and filtered-multi dimension neural network for improvement of reliability of cloud services. The knowledge-augmented temporal logic utilizes semantic extension to analyze dynamic attributes of entities and dynamic relationships between entities. The semantic extension provides knowledge supplements for logical reasoning. The temporal logic filter is implemented in the neural network framework to select sufficient attributes as input of hidden neurons. A novel reverse pattern tree data structure is also developed. The reverse pattern tree provides efficient insertion, search and compression functions for event pattern learning and recognitions. The filtered-multi dimension neural network provides more accuracy on multi-feature sequential event pattern predictions.

CHAPTER 2: LITERATURE SURVEY

2.1 Reliability in the Cloud

In large distributed systems, such as cloud computing systems, reliability is one of key characteristics in both system design and implementation. Cloud computing incorporates the computation ability of distributed systems and ease access of the Internet. Characteristics of cloud computing include service oriented, loose coupling, strong fault tolerant, business model, ease use, TCP/IP based, high security, and virtualization [4]. The fault tolerance is one of the most important features for cloud computing. The lack of error and fault handling coverage has been shown to be a drastic limit to dependability improvement [28]. Reliability of cloud computing depends on the ability of fault tolerance [27]. Services running in cloud usually run for weeks or even longer. During the runtime, there may be faults or errors happen in systems. If there is any fault happens, failure transactions usually will be roll back. Because there is almost no dependency between two transactions in cloud computing, failure transactions will not affect other transactions.

Faults in cloud computing can be categorized as provider-inner faults, provider-across faults, provider-user faults, and user-across faults [4]. If a fault happens in provider side and services are not urgent, this provider may restart services or start back-up services. If services are urgent or critical, the fault prevention is more important than fault recovery. When a fault occurs among providers, the transaction will be cancelled and return an error. The transaction will be redirected to other providers through load balancer. This type of faults does not occur very frequently. Between providers and users, the situation is more

complicated. There are so many factors can cause faults, such as network failure, browser crush, request time out or hacker attacks. When users are facing these types of faults, they usually resubmit their requests. However, if there is any key element involved in faults or errors, additional action may be needed to deal with system logs. If there is anomaly behavior occurred in faulted nodes, these nodes may need extra attention from cloud system protection and security point of view. Through the cloud platform, users not only connect with service providers but also share resources and activities with other users. In this situation, users manage their own critical resources. Unsafe security management configurations may cause unsafe access to critical resources. This is a critical issue in cloud computing systems. Service providers need to provide such functionality to help users to analyze their configurations.

Faults can also classified into categorizes according to features of faults [5]. Certain faults are caused by natural phenomena without human participation. This type of faults is natural faults, such as production defects. Natural faults also include internal and external faults. Internal faults are due to natural processes that cause physical deterioration. External faults are due to natural processes that originate outside the system boundaries and cause physical interference by penetrating the hardware boundary of the system or by entering via use interfaces. Other faults are all human-made faults. Human-made faults include malicious faults and nonmalicious faults. Malicious faults occur in either system development process or during directly use of system. Both of these types of malicious faults are with object to cause harm to the system. Nonmalicious faults are without malicious objectives. Nonmalicious faults can be further categorized into two classes:

nondeliberate faults and deliberate faults. Nondeliberate nonmalicious faults are due to mistakes occurs in the entire software life cycle. Developers, operators, maintainers can make mistake in any time. This type of faults is hard to prevent. Deliberate nonmalicious faults are caused by wrong or bad decisions. These bad decisions are made either accidentally or intended. Intended bad decisions are made because of lack of professional competence. Therefore, nonmalicious faults also fit into two types: accidental faults and incompetence faults. Malicious faults are created to fulfill malicious users' objectives. Malicious faults are categorized into two classes: malicious logic faults and intrusion attempts. Malicious logic faults can occur during the development processes, such as Trojan horses and logic bombs, and operation processes, such as viruses and worms. Intrusion attempts are performed by malicious users, who try to access confidential information or get unauthorized rights during the operation processes. Intrusion attempts can be logical and physical. In order to prevent and avoid system faults in cloud computing systems, above information is useful to clarify system design objectives.

Correct service is delivered when the service implements system functions [5]. A service failure is an event that causes delivered service cannot provide correct service. A service failure is a transition from correct service to incorrect service. Authors in [5] also categorized system failures into different categories: domain, detectability, consistency, and consequences. In domain category, failures contain content failure, early timing failure, late time failure, halt failure, and erratic failure. Every type of failures focuses on different aspects of system abnormal behaviors. In order to prevent system failures, we may need a unified framework for most of these types of system failures.

In traditional software reliability engineering, there are four main approaches to build a reliable software system. These four approaches are fault prevention, fault removal [30], fault tolerant, and fault forecasting [29]. However, in cloud computing environment, large-scale complex cloud applications only accept fault-prevention techniques and fault-removal techniques to develop fault-free software systems [7]. In paper [7], authors present a cloud application component ranking framework to build fault-tolerant cloud applications. The large scale cloud applications involve large number of components. Failures of these components affect reliability of cloud applications directly. The idea of presented framework in this paper is to find most reliable critical components in order to build reliable cloud applications. Based on the 80-20 rules, authors identify that the reliability of software system can greatly increase by eliminate small part of faults in the most important cloud application components. The paper also presents two ranking algorithms to identify significant components from the huge amount of cloud components and an optimal fault-tolerance strategy selection algorithm. Fault tolerance can increase the overall system reliability of cloud applications. One way to improve system reliability is to employ reliable components that provide equivalent functions to tolerate component failures. The redundantly devices, which help to maintain system functionality in the presence of failures, provide additional performance in their absence. In such situations, it is therefore necessary to employ metrics that take into account both system performance and reliability [26]. In order to tolerate faults by using redundant components, there are three well-known fault-tolerance strategies. The first one is Recovery Block [8]. Recovery Block is structure of redundant program blocks, where secondary blocks is activated when primary blocks fails. A recovery block fails only when all redundant blocks fail. The

second strategy is N-version Programming [9]. It is also known as multiversion programming, which provides multiple functionally equivalent modules. These modules generate result independently. The final result is selected according to majority voting. The third strategy is Parallel strategy. Similar to N-version Programming, modules in Parallel generate result independently. The first response of all results is selected as the final result. Component selection strategies use redundant components to form reliable cloud applications. Under these strategies, service providers need more resources to build services, which have lower cost efficiency. The paper [7] also proposes an optimal fault-tolerance strategy selection algorithm, which calculates the cost, response time, and the aggregated failure probability values of different fault-tolerance strategy candidates. The output of proposed algorithm is the strategy candidate with best failure probability performance. The proposed framework in [7] does not increase the reliability of each component of services. However, this framework increases the overall reliability of entire service through combinations of redundant service components. The drawback of this framework is service providers need more resources to build complete services. And furthermore, the more critical components a service has, the more redundant components are needed to tolerate faults.

Most cloud service providers deploy their services in large datacenters. All of services are running in virtual machines that reside in physical machines. There are usually multiple virtual machines running in one physical machine. When a virtual machine is initialized, the administrator or virtual machine monitoring system gets resources from a resource pool to build requested virtual machine [10]. In the paper [11], authors provide a

discussion on parametric sensitive analysis of availability of virtualized servers. The result of this analysis shows that host failures are the most important factors that affect mean time to failure of a virtual machine subsystem. Furthermore, the failure rate of applications is the major concern of capacity oriented availability. Both mean time to failure of virtual machine subsystems and capacity oriented availability are considered as a part of reliability of entire virtual system in a virtualized environment.

Reliability is the proper functioning of the system under the full range of conditions experienced in the field. In order to increase the reliability of systems, there must be some mechanisms in systems can either avoid the system failures and faults or adjust systems to prevent the more serious failures and faults. Of course, there is no system can ensure 100% reliability. System faults always happen in the entire computer systems, including cloud computing systems. There are researchers study architecture-based reliability prediction techniques, in order to increase reliability of computer systems [31, 32, 33, and 34]. The architecture-based reliability prediction tries to increase the reliability in the architecture design stage. In architecture-based reliability prediction design, the system usage profile modeling is one of approaches.

System usage can be described in terms of the expected sequences of system calls, which may influence the control flow throughout the system. In most of existing approaches, the system usage profile is encoded into transition probabilities between the states or scenarios of the system model [36, 37]. In [35], authors present a reliability modeling and prediction technique that considers the relevant architectural factors of software systems by explicitly modeling the system usage profile and execution

environment and automatically deriving component usage profiles. The transition probability is hard to get in the architecture design stage. However, it is easy to get statistical data in the runtime. And the system event logs record most of system events that include system fault related events. We can trace system faults through system event logs. There are always system critical events happened before system enter fault states. Therefore, if a system could predict system critical events, it can predict system faults before they really happen. Researchers dig into this problem from different aspects. Following study presents several techniques for system fault monitoring.

Some approaches are implemented with hardware support, such as [12]. Authors present a virtual lockstep implementation, which is software based, yet capable of using existing hardware features to enhance performance and fault detection capabilities. By modifying the KVM hypervisor to support virtual lockstep, the error detection is enhanced by verifying the state of the virtual processor at the deterministic VM exit boundaries. The replica virtual machine is used to accept the same inputs as the original virtual machine and provide out for error detection. The replica virtual machine maintains the same CPU state as the original virtual machine. Errors are detected by comparing both the outputs generated and the input types, which indicate calculation errors or significant divergences in execution. The experiment results show the virtual lockstep model introduces some runtime performance overhead to the system. The overhead rate is various based on the specific workload of virtual machines. The error detection of this virtual lockstep model only focuses on the fault execution of instructions in virtual machines, which ignores other errors and faults, such as errors of the instructions and application faults.

Some approaches focus on software and operating system level [13, 14, 38]. Authors of the paper [13] present a progress monitoring-based fault detection mechanism to detect and recover the driver VM from faults to enhance the reliability of the whole system. In hypervisor Xen architecture, an isolated driver model is introduced for I/O device virtualization. An isolated driver domain (IDD) is a special purpose guest domain for directly handling of I/O device accesses. In this reliability enhanced Xen, all the requests of guest domains are forwarded to their corresponding IDD. The IDD is build based on commodity operating system, such as Linux. Therefore, failure happened in the I/O device driver will not affect the entire system. In order to detect faults and errors, authors also present a detection module called Driver VM Monitor (DVM), which periodically detects the fault states of IDDs. The DVM is reside in the hypervisor and gets support from the virtual device drivers. When the fault detection function is invoked, the DVM logs the information of the I/O ring and physical IRQ (PIRQ). An I/O ring is a circular queue data structure containing descriptor information, is used to communicate between the front-end and back-end drivers. Front-end drivers are drivers reside in the VMs. Back-end drivers reside in IDDs and communicate with native device drivers. DVM mainly refers the I/O ring pointer information for fault decision, and the PIRQ is used for a more detailed inspection. If there is a fault occurred in an IDD, a recovery mechanism, called Driver VM Handoff (DVH), transparently redirects I/O requests and responses of the fault IDD to another IDD. The proposed monitoring module and recovery mechanism provide a fault detection and recovery on isolated device drivers.

Haibing et al. [14] introduce an Event-Based Polling model (sEBP), which uses existing system events to trigger a regular packet polling such that network interrupts are eliminated from the critical I/O paths in the virtual environment. In the sEBP architecture, events are retrieved by an event collector. An event manager is used to throttle the number of effective events out of the event poll. The event manager consists of three sub modules: rate controller, compensating timer, and cross-VM event sharing. The proposed model can be implemented in either guest OS in Virtual machine or hypervisor. The sEBP guest OS implementation collects system events in the guest VM. Another implementation, which sEBP is built in the hypervisor, collects VM_EXITs from various VMs as events. These two implementation methods fulfill two different administration goals.

Traditional techniques, such as heartbeat [43], have been frequently used to check the aliveness of physical and virtual machines. Statistical learning methods are also used to detect the system fault, such as [15, 16, 44, and 45]. Authors in [15] present a fault detection framework for virtualized environments. This framework consists of two phases: a training phase and a detecting phase. The training phase use historical data to train a Bayes classifier. The train data is collect from multiple levels of virtualized environments. In the second phase, the trained model is used to detect runtime system faults. In such virtualized environment, different levels generate different system faults. The proposed framework concentrates on three levels; application server level, operation system level, and virtual machine monitor level. The application server level is also considered as the virtual machine level. In [15], the application server level extractor is a filter resides in the application to collect application behaviors. The OS level extractor monitors OS events

and faults. The VMM level extractor collects adjustment events of virtual machines. Authors utilize the Bayes classifier on features that are extracted from multiple levels. The trained model is used in prediction, which is not mentioned in the paper.

Authors in [16] detect aging phenomenon by conducting experiments in physical and virtual machines and identify the differences between the two, and propose a feature code-based methodology for failure prediction through system call. The aging problem is caused by a large number of repeated executions. Authors also define aging rate as the metric of decreasing trend which quantifies the variation of system resource usage. The proposed prototype is implemented in the VMM layer to predict the rejuvenation time.

2.2 Temporal Logic and Policy Analysis

Temporal logics have experienced rapid development in recent years. Various properties for temporal logics' complexity and axiomatizations are studied [18, 19 and 20]. Logical expression capability makes temporal logic a good tool for system specification and verification. Recently, temporal logics are used more in reasoning and planning as well [21, 22, and 23], especially in policy specification reasoning and analysis [24]. In distributed environments, one entity may carry multiple attributes and these attributes can have different definitions in different domains. The complexity of an information domain becomes a barrier for specification and verification of policies. There are two major categories of temporal logics that can be used to analyze temporal attributes. One is linear-time temporal logics; the other is branching-time temporal logics. In the first category [25 and 46], information is represented as constraints. In [25], authors implement a Dynamic Linear Temporal Logic (DLTL) to specify and verify systems with communicating agents

and interaction protocols. Semantic facts of agent communication are specified by means of rules and constraints. In [46], authors describe a logical framework for Temporal Action Logic (TAL) that specifies and verifies interacting systems. This framework provides a simple formalization of communicative actions in terms of their effects and preconditions and the specification of an interaction protocol by means of temporal constraints. Another temporal logic [47] achieves effectiveness and simplicity through reduction of information from information domains. Authors present an A-LTL that inherits some properties from Linear-time Temporal Logic (LTL), including constraints. Interval Temporal Logic (ITL) [48, 49] is another linear temporal logic working over finite time intervals. The Propositional Interval Temporal Logic (PITL) [50, 51] is an extended Interval Temporal Logic, which considers semantic information through past operators. However, if some information elements cannot be expressed by logic operators, the accuracy of reasoning may be compromised. In [52], a Fuzzy Temporal Logic is proposed. The fuzzy temporal constraints are used for simple cases, where constraints are composed in a single interval. Constraints usually play as a supplement to logical reasoning, which contains limiting conditions from an information domain. Borrowing from this idea, I propose a semantic extension as an addition to temporal logic so that hidden and implicit relationships can be expressed and incorporated in temporal analysis. Meanwhile, a balanced point of time complexity and space complexity can be achieved through proper usage of this semantic extension.

Research in policy conflict analysis has attracted growing interest recently as autonomous and automatic system management has become popular. Dynamic policy

analysis also has started to be studied recently. Logic languages [53, 54] are widely used in this field. Temporal logic is widely used in different types of policy analysis frameworks. For example, First-order Temporal Policy-analysis Logic (FTPL) [53] is used to check whether a SPKI policy state satisfies a property specified in FTPL. This property check can be applied to static properties and static policies, which is insufficient for collaboration activities. In [54], Event Calculus is implemented in a logic-based policy analysis framework to represent and perform reasoning about inconstant properties of a domain regulated by policies. However, this framework can only statically analyze policies when it monitors runtime policies. Other policy analysis mechanisms [55, 48, and 56] also focus on static conflict analysis.

Dynamic policy analysis is growing in recent years. In [17], authors present an approach implemented in their DiffServ QoS management platform to analyze policy conflict through Event Calculus. They argue that application-specific conflicts are dynamic and can only be determined at run-time, because such conflicts depend upon current status of the system. They also illustrate several types of potential conflicts that may arise in dynamic resource management for QoS support. An event-driven conflict detection mechanism is introduced in [58], which uses a conflict database to store all possible conflicts. If there is an event that may cause a conflict, this mechanism will check corresponding database entries. These dynamic conflict analysis mechanisms can monitor policy sets during run-time, but they cannot trace implicit attributes in policies and dynamic relationships among these attributes. In [59], authors use a set of conflict-related Boolean rules to verify policies in order to discover and resolve IPSec policy conflicts. In [60],

authors propose a unified model to represent and encode QoS policies for efficient conflict analysis. In [61], authors propose a logical reasoning framework for policy analysis in mobile social network. This framework focuses on the analysis of geological location information, which also changes over time. However, it cannot analyze policies from other domains with other types of dynamic information. So I propose a general framework that can be applied to most types of dynamic policy analysis.

Meanwhile, logical agents have been studied for decades and implemented in many different research fields. In [62], authors present an agent-based conceptual and computational model of consumer decision-making based on culture, personality and human needs. These needs are supplied in a knowledge base to drive the consumer toward a specific product. In [63], authors present an Agent-Based Modeling (ABM) as a viable tool to account for the interaction of local and environmental factors to determine organizational success. Again, a comprehensive knowledge base is constructed to help decision-making. In [64], authors describe an analysis and simulation of meta-reasoning processes using an agent architecture for strategic reasoning in naval planning. In these papers, a knowledge-augmented logical agent can not only make decisions but also perform logical reasoning and data analysis to support decision-making. Therefore, I will also incorporate knowledge into logical reasoning and a knowledge base into our policy analysis framework to help process implicit attributes and relationships.

2.3 Event Learning and Prediction

System event monitoring collects statistical data of system events. Through machine learning techniques, we can find some patterns that always appear when system

faults occur. Statistical data is used in mining and detecting fault patterns, [39, 40, 41, and 42,] mined multivariate time series by recognizing them as weighted graphs to monitor the graph sequences for failure detection. System critical event prevention is possible through prediction of system critical event. If there is a high possibility of system critical events, cloud computing systems can try to avoid system critical events. There are a lot of event pattern learning and prediction frameworks. But Agrawal and Srikant proposed the first sequential pattern mining problem in 1995 [57]. Agrawal also presents an Apriori-based method which is Generalized Sequential Pattern algorithm (GSP) in [82]. The GSP algorithm screens all length-1 candidates in the database. Those sequences with support less than the minimum support are filtered out. Then the algorithm screens the database length-k times to collect support count for each candidate and generates candidate length-(k+1) sequences from length-k frequent sequences using Apriori. GSP algorithm generates a huge set of candidate sequences in multiple database scans which is inefficient for large databases.

Cloud computing platforms provide various services, such as infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS) and etc. Most of these services are managed through configurations and policies. When a user tries to use two services at the same time, user's activities have to fulfill both requirements to be accepted. Conflicts happen during collaboration of services, integration of services and separation of services. Conflicts of policies or other management requirements prevent services to provide correct functionalities to users. Furthermore, services may provide wrong result for users according to conflicted management policies. Therefore,

management policy conflict elimination is one aspect of maintain reliability of cloud computing platform. In order to prevent conflicts in these scenarios, conflict analysis techniques have been studied.

CHAPTER 3: RESEARCH PLAN OVERVIEW

3.1 An Autonomous Agent Architecture for Reliability

In cloud computing, service providers always want to provide reliable services to customers or service consumers. However, there are obstacles between service providers and consumers. Customers need customized services with various configurations, these customizations and configurations make service providers hard to control the stability of their service, especially from different management domains. Configuration in services are usually expressed as configuration policies. Therefore, Conflicts between service providers and consumers is one obstacle that affects the reliability of services in cloud computing environment.

One of them is policy conflicts when consumers try to integrate multiple services from different domains. When services from one provider may not fulfill consumers' needs, Service consumers have to integrate multiple services from multiple service providers, who are in different management domains. These service providers set up their own policies including management policies, control policies, privacy policies, security policies, etc. Policies from different service providers may not compatible with each other. Therefore, we need a policy analysis mechanism to find out incompatible parts of policies during the service integration. In the agent architecture, knowledge base will contain different domain information and the mapping relation of domain objects.

Another obstacle is how to detect abnormal events and predict them during the service collaboration. In cloud computing environment, although each virtual machine or service is independent for users, the entire physical environment is integrated. When users are using services that are running on virtual machines or virtual servers, resources are usually allocated through the virtual machine monitors. Virtual machine monitors do not only allocate resources but also monitor status of virtual machines and system events of virtual machines. System events in virtual machines are system calls, errors, or failures. These system critical events reflect the status of virtual machines. Sometimes, these system events also affect the reliability of entire virtualized environment. When one virtual machine crashes, physical machine needs computing power and storage resource to restore crashed virtual machine. The restore process does need extra resources rather than running a stable virtual machine. In order to increase the reliability during the integration, there must be a mechanism to predict system critical events and prevent them. This mechanism can be implemented into an agent architecture to fulfill needs of autonomous and efficiency.

To overcome these obstacles, I designed an autonomous agent framework (Figure 1), which is a customized agent architecture including policy analysis and service critical event prediction for cloud computing services. This framework monitors service status through service control policy monitoring and service event monitoring. Control policies and service event logs are normalized according to domain information in knowledge base. Policy and event analysis is done by inference engine. Knowledge base contains policy model and event pattern statistical information. This framework also provides suggestions according to the result of analysis. Following sections describe major components in this

framework. Chapter 4 discusses the policy monitoring and analysis components. Chapter 5 discusses the non-intrusion Log processing components.

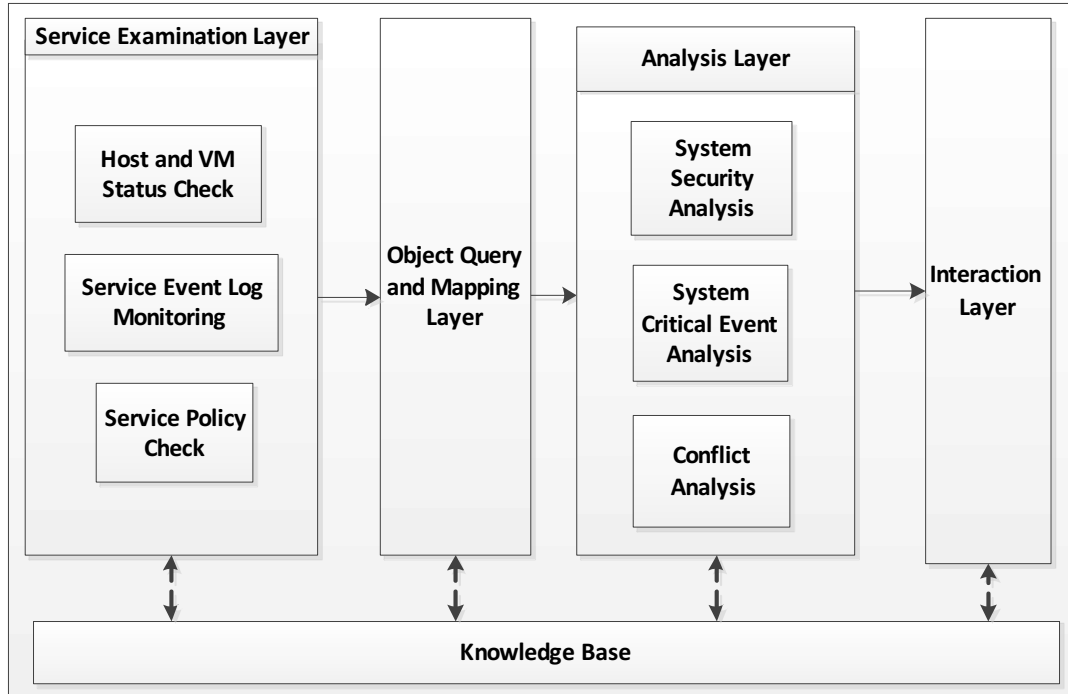


Figure 3.1. Autonomous Agent Architecture for Reliability Monitoring

CHAPTER 4: SERVICE POLICY ANALYSIS

4.1 Policy Modeling

4.1.1 General Policy Model

Service collaboration is a clear trend for cloud applications and services over the Internet. Guarantee for Quality of Service (QoS) is an important yet difficult task to accomplish among collaborating applications and services. However, different services have different control and management requirements. When customers try to use multiple services, the control requirements have to be consistent. For example, a real-time HD video service requires a large bandwidth for any synchronization or collaboration, and a messaging service requires low delay and secure transmission for collaboration. These two services may need to be integrated and work concurrently in a multimedia application. In order to manage complex requirements between multiple services and service consumers, policy-based management can be applied and these requirements can be represented in policies. Policy-based management is an administrative approach to manage system usage and its governance rules within an information domain. More and more systems have adopted this policy-based management approach. In a collaborating services environment, a policy domain (domain hereafter) is a collection of elements and services administered in a coordinated fashion [1].

Collaborating services can support interactions and coordination between service providers and individual services, as well as service providers and service consumers. Different service providers can share their resources and build new services based on

existing services. For example, in Figure 4.1.a, domain A contains two services: financial service *A* and data service *D*. Service *A* requires data service *D* to provide enough throughputs. Another domain B contains two services: message processing service *B* and data service *D*. Service *B* needs data service *D* to respond to every request within a certain time limit.

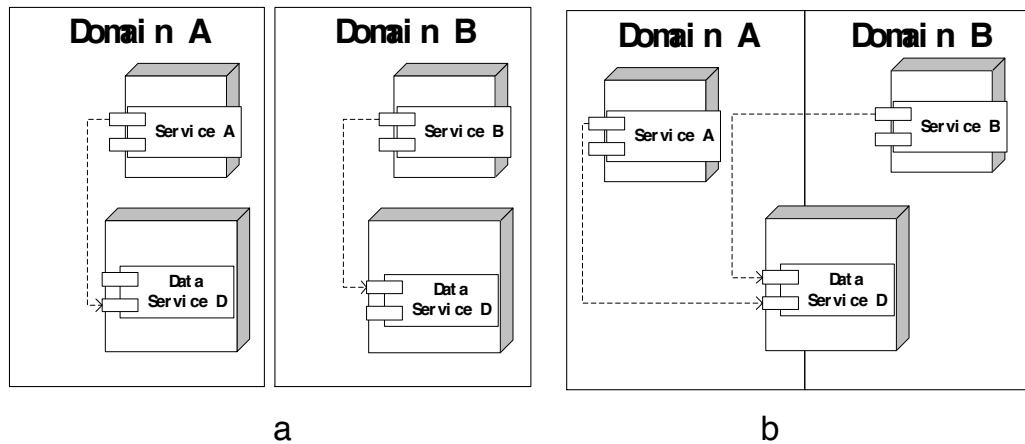


Figure 4.1 Service in Different Domain

When these two domains collaborate, they share the same data service *D* (as illustrated Figure 4.1.b). At this point, data service *D* has two policies from service *A* and service *B* respectively. However, different management requirements from these two services are reflected in different policies in a policy-based management environment. These requirements may conflict with each other. These conflicts of requirements are usually reflected in policy rule conflicts. For example, before domain A and domain B collaborate, they have their own policies to control services and have their own data services. I call the policy in domain A “Policy 1”, and the policy in domain B “Policy 2”. In “Policy 1”, data service *D* has to provide enough throughputs for service *A* and the maximum throughput

is 2MB/s. This maximum throughput value is adjustable according to the number of page views in service *A*. In “Policy 2”, data service *D* has to respond to any request from service *B* within 100 time units (0.1 second). Page view on service *A* is a dynamic and hidden factor, which affects requirements from service *A*. Before collaboration, data service *D* provides certain throughput for service *A* only. Therefore, service *D* can use all of its capacity to serve *A*.

In a policy-based management system, a policy is a statement that describes what entities can do and how these actions can be performed. In other words, a policy describes several actions and information about these actions. For each action, there is an executor or a type of executors, a target or a type of targets, and certain constraints, which constrain and describe certain aspects of the action. Each executor or target is represented by a set of attributes. An attribute is a characteristic of an entity. In most policy languages, users can define very comprehensive policies containing different actions and their constraints. Although one policy is enforced as an entirety, it can be decomposed into policy segments for analysis. I assume that one segment describes one complete action with one executor, one target, and its context (in the form of constraints). Therefore, a policy segment is represented as a tetrad (Executor, Target, Action, and Context). Previously, a policy model for collaborative services is introduced in [70]. That model is used to represent policies formally. There is no domain information in that policy model, which makes policies independent with their domains. However, in the policy analysis, we have to incorporate domain information into the analysis process. Therefore, I extend the policy model to make it more general and make it having a dynamic capability.

Definition 1: Attribute (α) is a piece of information representing a characteristic of an entity. For example: A user's ID number is an attribute.

Definition 2: Entity (E) is a collection of attributes describing a complete element in an information domain. $E = \{\alpha_1, a_2, a_3 \dots a_n\}$;

This set of attributes may be independent or interrelated. An entity can also be a set of entities. I use capital letters to represent entities. A user is an entity in an information domain; a file is also an entity in an information domain.

Definition 3: Relationship (R) is represented by a predicate of the Cartesian product of two entities: $R = P(E \times E')$.

If R is a relationship between entity A and D , then R is a predicate of $A \times D$. $P()$ is a predicate. The value of R represents whether the relationship holds. The Cartesian product of A and D is the set of all ordered pair $(val(\alpha), val(\beta))$, where $\alpha \in A$ AND $\beta \in D$, and $val(\alpha)$ and $val(\beta)$ are corresponding values of a and β . The value of predicate P is determined by selected pairs $(val(\alpha), val(\beta))$ in the entire ordered pair set. For example, if attributes a and β contain two numerical values, the relationship R is hold when $val(\alpha) = val(\beta)$ is hold. The relationship is *Equal* of two numerical values.

$$Relation = Equal(\{\alpha\} \times \{\beta\}) = true, iff \{(\alpha, \beta) | \alpha \in A \wedge \beta \in D \wedge (val(\alpha) = val(\beta))\}$$

Definition 4: Action is a function on a relationship between related entities with constraints. An action is denoted as $action(R, \{x\})$.

In an action, the input is a relationship between two entities and constraints effective on the relationship. The output is an altered relationship. The symbol R represents a relationship between two entities. The symbol $\{x\}$ is a constraint set for this action. This constraint set $\{x\}$ contains zero or more constraints. Different actions concentrate on different relationships of between entities. For example, $action()$ is an action that A can change the value of attribute β of entity D . It is represented as $action(R, \{x\})$. The relationship R between A and D indicates there is one attribute a of entity A is equal to one attribute β of entity D ($R=Equal(A \times D)$). The constraint x is the value range of attribute a . When this action is performed, the entity A changes the value of its attribute a within the range that constraint x states. In order to maintain the true value of relationship R , the attribute β also needs to be changed to the new value of attribute a .

Definition 5: Executor is the source entity of an action, which is the E in the corresponding relationship. $E=Executor=\{\alpha\}$. Attributes in an Executor entity initiates changes to break relationships.

Definition 6 Target is the recipient entity of an action, which is the E' in the corresponding relationship. $E'=Target=\{\alpha\}$; Attributes in a Target entity respond to changes to maintain the relationship.

Definition 7: Constraint is restrictive information or conditions on entities and actions.

Constraints of a policy usually include restrictive information or conditions from the system environments or policies. The constraint restricts the executor, target, and/or action. In this policy model, I separate the constraints into two types: one type of constraints

restricts actions, and another type of constraints restricts entities. I consider the second part of constraints as a part of the Context.

Definition 8: Context in a policy segment includes constraints on entities and environmental constraints.

A policy segment describes a single action, which contains only one executor, target, action, and context. In each policy segment, an action represents a function on a relationship between its executor and target. Therefore, actions work on mappings of attributes between executors and targets, which are legitimate entities in a policy domain. Executors and targets consist of sets of attributes.

Definition 9: Policy Segment is the smallest functional policy unit in a policy. A segment is a tetrad: $Segment = (Executor, Target, Action(R, \{x\}), Context)$.

The above definitions are common knowledge. I define them in order to formally present a policy in the following sections.

4.1.2 Policy Representation Using Temporal Logic

“The primary feature of a logic theory is its order, which defines the domain of all formulae described by the logic.” [64]. Propositional logic is based on a set of elementary facts connected by a set of logical operators. It indicates a Boolean value set. First-order logic [71] is an extension of propositional logic. Temporal logic assumes that facts hold at particular time periods, or before or after certain time points, and these time periods and points are ordered [72]. In order to incorporate the domain information with the policy analysis, I proposed a semantic extension to the temporal logic. So implicit relationships

and other hidden information can be expressed and incorporated in the temporal analysis. I can use temporal logic to express policies. The described policies in the previous section can be expressed as the following logical expressions.

“Policy 1” can be expressed as:

$$\{A, D, requestthroughput(\Theta, \alpha), [permit(A, D, requestthroughput(\Theta, \alpha)) \\ = true, iff A \in DomainA \wedge D \in DomainA \wedge \alpha < 2000KB/s]\}$$

In this logical expression, A is the executor; D is the target; $requestthroughput(R, \alpha)$ is the action that executor A performs on target D ; Θ is the relationship between executor service A and target service D ; α is the number that A needs to request. In this policy, the relationship Θ means there are two attributes in A and D is equal to each other

$$[permit(A, D, requestthroughput(\Theta, \alpha)) = true, iff A \in DomainA \wedge D \\ \in DomainA \wedge 1000KB/s\alpha < 2000KB/s]$$

is the context within “Policy 1”. Predicate $permit(A, D, requestthroughput(\Theta, \alpha))$ is true when A is allowed to perform action $requestthroughput(\Theta, \alpha)$ on D . Context information is placed in square brackets “[]”. According to “Policy 1”, if service A and data service D both belong to “ $DomainA$ ”, when service A needs more bandwidth, service A can request certain amount of throughput denoted as a from data service D for itself (α must be smaller than 2000KB/s).

“ Policy 2” can be expressed as:

$$\{B, D, \text{requiredresponsetime}(\Phi, 100), [\text{permit}(B, D, \text{requiredresponsetime}(\Phi, 100)) \\ = \text{true}, \text{iff } B \in \text{Domain}B \wedge D \in \text{Domain}B \wedge (\text{Response}_D < \text{Response}_B)]\}$$

Here, B is the executor; D is the target. The executor service B can perform action $\text{requiredresponsetime}(\Phi, 100)$ on the target service D . This action is used to set the response time requirement in service D . Because the parameter in this action is a constant value, this action reflects that the requirement of service B always stays at 100 milliseconds. The relationship Φ represents Response_time in D has to be smaller than the response time requirement from B .

$$[\text{permit}(B, D, \text{requiredresponsetime}(\Phi, 100)) = \text{true}, \text{iff } B \in \text{Domain}B \wedge D \\ \in \text{Domain}B \wedge (\text{Response}_D < \text{Response}_B)]$$

is the constraint in “Policy 2”. In this policy, there is only one response time requirement between service B and data service D . According to this policy, the response time requirement is a fixed value. In other words, unless data service D cannot provide response within the required time limit, there is no violation against with this policy. The required response time is denoted as Response_B ; the response time in data service D is denoted as Response_D . Service B cannot change the response time setting in data service D . Predicate $\text{permit}()$ evaluates whether executor B can perform action $\text{requiredresponsetime}(\Phi, 100)$ on target D .

In the above logical expressions, there is no time or time-dependent information on executors, targets, actions or constraints. If there is time-dependent information in a policy, it is difficult to be represented by first-order logic only. For example, in a situation such as

“service A is using data service D”, the term “using” is the critical part, which expresses the action being preformed. Temporal logic can be used to represent time-dependent situations. It has been broadly used to cover temporal information in many logical analysis methods. As an extension of first-order logic, temporal logic can also express policies. Temporal logic adds time elements into expressions without changing the semantics. Then, the two policy examples can be expressed as follows.

Policy 1:

$$\{A, D, requestthroughput(\Theta, \alpha), [HoldsAt(permit(A, D, requestthroughput(\Theta, \alpha)), t) \\ = true, iff A \in DomainA \wedge D \in DomainA \wedge 1000KB/s < \alpha < 2000KB/s]\}$$

Time information is added to “Policy 1”. This is a constraint that does not affect the format of the general policy model. The time information becomes another dimension that is denoted as t in logical expression. In this expression, if A and D belong to $DomainA$ at time t and $1,000KB/s < \alpha < 2,000KB/s$, the predicate $permit()$ holds true. Figure 2 (a) illustrates the range of throughput that can be requested by A .

Policy 2:

$$\{B, D, requiredresponsetime(\Phi, 100), [HoldsAt(permit(B, D, requiredresponsetime(\Phi, 100)), t) \\ = true, iff B \in DomainB \wedge D \in DomainB \wedge (Response_D < Response_B)]\}$$

In this expression, if B and D belong to $DomainB$ at time t , the predicate $permit()$ holds true.

According to the above logical expressions, there is no conflict between these two policies. In Figure 4.2(a), D 's throughput for A will change between 1,000KB/s and 2,000KB/s; in Figure 4.2(b), D 's response time for B will be a constant yet smaller than B 's requirement. These two policies are defined on different attributes of data service D .

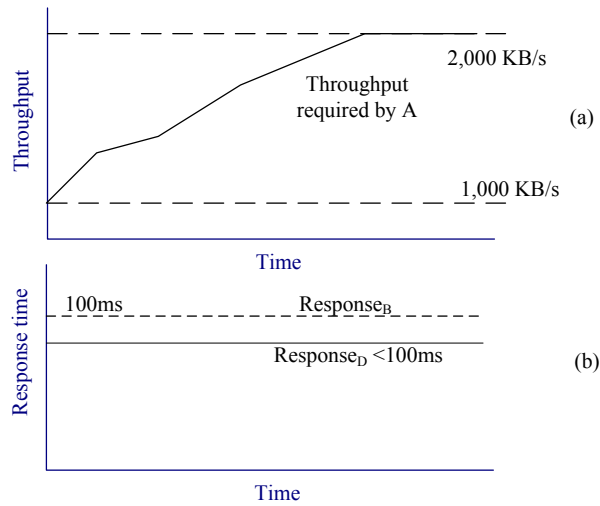


Figure 4.2 Throughput and Response Time for Data Service D

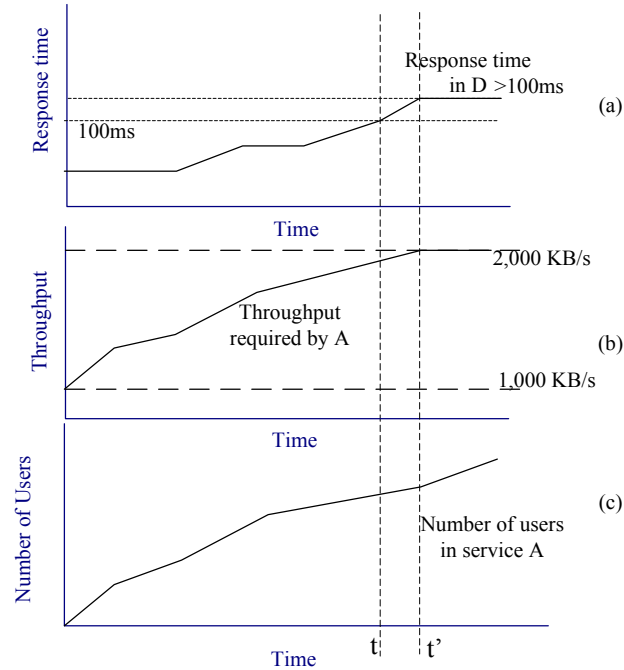


Figure 4.3 Relationships between Two Policies over Data Service *D*

It seems that there is no conflict between these two policies. However, if I take a closer look at the environment of this collaboration, I discover that service *A* cannot always increase its throughput requirement. Otherwise, data service *D* cannot respond in time to service *B*. Figure 4.3 shows this situation with these two policies. In this diagram, when the number of users for service *A* increases (as shown in Figure 4.3 (c)), the throughput requested by service *A* also increases until the throughput reaches 2,000 KB/s (as shown in Figure 4.3(b)). Along with the increase of throughput requirement, data service *D* has to use more resources to support service *A*. At the same time, data service *D* cannot avoid increasing its response time for service *B*, because it does not have enough resources for service *B*. As shown in Figure 4.3 (a), at time t , the response time of data service *D* reaches the upper limit of the requirement from service *B*, but it will still increase. At time t' the

response time has already exceeded the requirement of service D . The conflict between these two policies is caused by hidden information. The hidden information does not shown in policies explicitly. Therefore, I introduce a semantic extension that contains domain and environment information to provide extra support for logical reasoning.

4.2 Knowledge-Augmented Temporal Logic

4.2.1 Semantic Extension

Logical agent is useful in policy conflict analysis, since it can provide automatic and autonomous analysis. There are four major components in an agent architecture: inference engine, knowledge base, sensor, and actuator. Semantic extension is an extended part of the knowledge base. A semantic extension contains attributes, relationships and dynamic constraints among attributes and relationships for an information domain. There is not only information changing along with others, but also information changing over time. This dynamic information imposes complications on logical analysis, and a traditional knowledge base is not enough for logical reasoning in policy conflict analysis with dynamic attributes and relationships. As a part of information, certain type of attributes that changes overtime, I define this type of attributes as dynamic attributes, such as temperature, throughput, and response time. The semantic extension provides dynamic information and supports logical reasoning. Semantic extension is a formal representation of related information abstracted from an information domain, which includes attributes, entities, relationships and constraints. Relationship is an important part of a semantic extension, which works on a Cartesian product of two entities.

$$\Theta = \text{Equal}(\{a\} \times \{\beta\}) = \text{true}$$

$$\text{iff } \{(a, \beta) \mid a \in A \wedge \beta \in D \wedge (\text{val}(a) = \text{val}(\beta))\}$$

If Θ is a relationship between *Service A* and *Data Service D*, Θ is a predicate on $A \times D$. A represents the set of attributes for *Service A*; D represents the set of attributes for *Data Service D*. *Equal* is the predicate. The value of Θ is whether the relationship holds.

For example, during the collaboration between *DomainA* and *DomainB*, A and D have a relationship: relationship Θ (A 's throughput requirement equals to D 's throughput provided for A); B and D have a relationship: relationship Φ (D 's real response time is less than B 's response time requirement). Relationship Θ can be expressed as the following form.

$$\Theta = \text{Equal}(A \times D) = \text{Equal}(A.\text{requiredthroughput}, D.\text{throughput})$$

$$\text{iff } (A.\text{requiredthroughput} \in A \wedge D.\text{throughput} \in D \wedge A \in \text{DomainA} \wedge D \in \text{DomainA})$$

In this logical expression, relationship Θ is consisted of predicate on two attributes: *A.requiredthroughput* and *D.throughput*. These two attribute belong to entities A and D respectively. In Policy 1, service A can change its throughput requirement, which has to be fulfilled by *Data Service D*. According to this policy, the attribute *throughput* in D has to be equal to the attribute *requiredthroughput* in A under certain conditions. I use *Equal(AxD)* to represent this requirement from Policy 1. Since the relationship Θ is consisted of a predicate over two numeric attributes, there are three situations *Equal(AxD)*, *Larger(AxD)* and *Smaller(AxD)* between two attributes. *Equal(AxD)* can be defined as the following form:

$$\begin{aligned}
& Equal(A \times D) = HoldsAt((A.requiredthroughput = D.throughput \\
& iff(\forall A.requiredthroughput, D.throughput \mid A.requiredthroughput \in A \wedge D.throughput \in D \\
& \wedge (1000KB/s < A.requiredthroughput < 2000KB/s) \\
& \wedge A \in DomainA \wedge D \in DomainA \wedge (A.requiredthroughput = D.throughput))
\end{aligned}$$

The $Equal(A \times D)$ represents a relationship with some constraints between two entities A and D . Under constraint $A.requiredthroughput = D.throughput$, relationship Θ becomes the relationship $Equal$. Constraint of a relationship will be discussed in section 4.2.

Relationship Φ is between *Service B* and *Data Service D*. D should have a smaller response time than the requirement from B , which means the required response time from B is larger than the response time in D .

$$\begin{aligned}
& \Phi = Larger(B \times D) = Larger(B.required_responsetime, D.responsetime) \\
& iff(B \in DomainB \wedge D \in DomainB)
\end{aligned}$$

In this logical expression, relationship Φ 's value is also consisted of a predicate over two attributes: $B.required_responsetime$ and $D.responsetime$. In Policy 2, service B has a response time requirement, which is a fixed value. This value is the maximum response that can be accepted by service B . Therefore, the response time in *Data Service D* must be less than this value. The requirement of this policy can be defined in the following logical expression:

$$\begin{aligned}
& Larger(B \times D) \\
& = HoldsAt(B.required.responsetime \\
& > D.responsetime, t) \text{ iff } \forall B.required_responsetime, D.responsetime \mid B.required_responsetime \\
& \in B \wedge D.responsetime \in D \wedge B \in DomainB \wedge D \in DomainB \wedge B.required_responsetime \\
& > D.responsetime
\end{aligned}$$

The relationship *Larger* is built on two attributes *B.required_responsetime* and *D.responsetime*. This relationship is a specific situation of relationship Φ with a constraint. Only when the constraint is hold, the relationship Φ is *Larger*.

When two relationships *Equal* and *Larger* are held, policies can be enforced on *A*, *D* and *B*.

The following temporal logic expressions illustrate *A*'s attributes to reflect possible attribute changes in the above relationships:

$$\begin{aligned} & \forall t, T < t \wedge \text{increased}(A.\text{requiredthroughput}, t) \wedge (1000 < A.\text{requiredthroughput} < 2000) \\ & \Rightarrow \text{HoldsAt}(\Theta = \text{Equal}, t) \\ & \Rightarrow \text{HoldsAt}(\text{permit}(A, D, \text{requestthroughput}(\Theta, A.\text{requiredthroughput})), t) \end{aligned}$$

In this logical expression, $\text{increased}(A.\text{requiredthroughput}, t)$ is a predicate, which means attribute *A.requiredthroughput* is increased at time *t*. After the increase, the *A.requiredthroughput* is still smaller than 2000. Thus relationship Θ can still be *Equal*. So I should permit the action $\text{requestthroughput}(\Theta, A.\text{requiredthroughput})$ to increase *D.throughput*. The predicate $\text{Permit}(A, D, \text{requestthroughput}(\Theta, A.\text{requiredthroughput}))$ is true when *A* can perform action $\text{requestthroughput}(\Theta, A.\text{requiredthroughput})$ on *D*.

$$\begin{aligned} & \forall t, T < t \wedge (B.\text{required_responsetime} > D.\text{responsetime}) \wedge (B.\text{required_responsetime} = 100) \\ & \Rightarrow \text{HoldsAt}(\Phi = \text{Larger}, t) \\ & \Rightarrow \text{HoldsAt}(\text{permit}(B, D, \text{requiredresponsetime}(\Phi, 100)), t) \end{aligned}$$

In this logical expression, *B.required_responsetime* is larger than *D.responsetime*, and *B.required_responsetime* equals to 100. At time *t*, relationship Φ is *Larger*. Then at time *t*, the predicate $\text{permit}(B, D, \text{requiredresponsetime}(\Phi, 100))$ can hold.

In service collaboration, certain relationships are implied in policies to control information and resource sharing. A relationship between two entities is a relationship between two attributes or two attribute sets from these two entities. However, certain attributes will affect relationships, and some of them will affect relationships indirectly and implicitly. So certain constraints need to be established for this type of effects.

Definition 10: When an attribute affects a relationship and makes it change I call this attribute an explicit attribute. Explicit attributes define a set of attributes that can initiate certain changes to a relationship. The superscript in a logical expression denotes an explicit attribute.

Definition 11: When an attribute is affected by a relationship, I call this attribute an implicit attribute. Implicit attributes define a set of attributes that are derived from this relationship, and these attributes are affected by a change to this relationship. Implicit attributes are denoted as suffixes.

For example, $\Theta^{A.requiredthroughput}_{D.throughput}$ indicates that if the attribute $A.requiredthroughput$ changes, the relationship Θ will not be held. In order to keep this relationship, the attribute $D.throughput$ will have to change. If $A.requiredthroughput$ increases, in order to keep the relationship, the throughput in D will also need to be increased, and the response time in D will have to increase as well. Figure 4.4 shows the difference and relationship between explicit and implicit attributes.

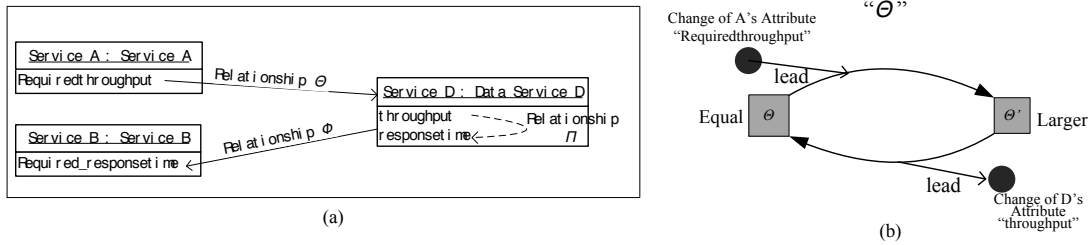


Figure 4.4 Relationships and Constraints

In Figure 4.4 (a), if the explicit attribute *requiredthroughput* changes, this change will lead to a change on relationship Θ . For example, relationship Θ changes from *Equal* (Θ) to *Larger* (Θ'). According to Policy 1, the value of *A.requiredthroughput* has to be the same as the value of *D.throughput*. In order to hold the requirement of Policy 1, the relationship Θ' has to be changed back to the relationship Θ . The attribute *D.throughput* has to be changed to meet the required value of *A.requiredthroughput* (as shown in Figure 4.4 (b)). The lead arrows in relationship Θ are solid; this means relationship Θ and its explicit and implicit attributes are explicitly illustrated in the policy. Attribute *D.throughput* leads to the relationship change on Π and then the relationship Π affects *D.responsetime*. The arrow in relationship Π is dotted lines, which means relationship Π is implicit in policies. Relationship Π has a *Balance* value, which means *D.throughput* and *D.responsetime* are related. When one falls, the other will rise. Finally, the attribute *D.responsetime* leads to the change of relationship Φ . I use a logical expression to represent explicit and implicit attributes, and store them in the semantic extension. Once this logical expression is retrieved from the semantic extension in the knowledge base, it can help track these attributes' updates and derived changes.

$$\begin{aligned}
& \forall t, T < t \wedge \text{increased}(A.\text{requiredthroughput}, t) \wedge (1000 < A.\text{requiredthroughput} < 2000) \\
& \wedge \text{HoldsAt}(\Theta_{D.\text{throughput}}^{A.\text{requiredthroughput}} = \text{Equal}_{D.\text{throughput}}^{A.\text{requiredthroughput}}, t) \\
& \wedge \text{HoldsAt}(\Pi_{D.\text{responsetime}}^{D.\text{throughput}} = \text{Balance}_{D.\text{responsetime}}^{D.\text{throughput}}, t) \\
& \wedge \text{HoldsAt}(\Phi_{D.\text{responsetime}}^{B.\text{required_responsetime}} = \text{Larger}_{D.\text{responsetime}}^{B.\text{required_responsetime}}, t) \\
& \Rightarrow \text{HoldsAt}(\text{permit}(A, D, \text{request}(A.\text{requiredthroughput})), t)
\end{aligned}$$

In the above logical expression, if action $\text{request}(\Theta, A.\text{requiredthroughput})$ holds, and the requested throughput is less than 2,000 KB/s, relationship Θ , Π and Φ hold and then action $\text{request}()$ is permitted. I denote the attribute $A.\text{requiredthroughput}$, which is an explicit attribute for relationship Θ , as a superscript and the implicit attribute $D.\text{responsetime}$ as a suffix.

4.2.2 Relationship and Entity

A relationship may not only be affected by attributes but also be affected by constraints. In an information domain, relationships connect different entities (e.g. services). Because an entity is represented by a set of attributes, relationships also connect different attributes. In Figure 4.5, *Service A* and *Service B* together with *Data Service D* have two relationships (implied in policies). *Service A*, *B* and *D* have three attributes respectively. Relationship Θ connects attribute $\text{requiredthroughput}$ in *Service A* and *Data Service D*. If there is a constraint on attribute $\text{requiredthroughput}$ in *Service A*, relationship Θ will be affected only when this constraint ($1,000 \text{ KB/s} < \text{throughput} < 2,000 \text{ KB/s}$ in this case) is satisfied. If this constraint changes over time, I call this constraint a “dynamic constraint”. Dynamic constraints are very important in an information domain because these constraints usually control the connection between different entities. In addition, attribute $\text{requiredthroughput}$

is an explicit attribute for relationship Θ , attribute *responsetime* is an implicit attribute for relationship Θ , and attribute *responsetime* is also an explicit attribute for relationship Φ . Therefore, if attribute *requiredthroughput* changes, it will affect relationship Θ and then relationship Θ will affect attribute *responsetime*. Attribute *responsetime* will eventually affect relationship Φ . If the results of these two changes are inconsistent, there will be a conflict (conflict of duty).

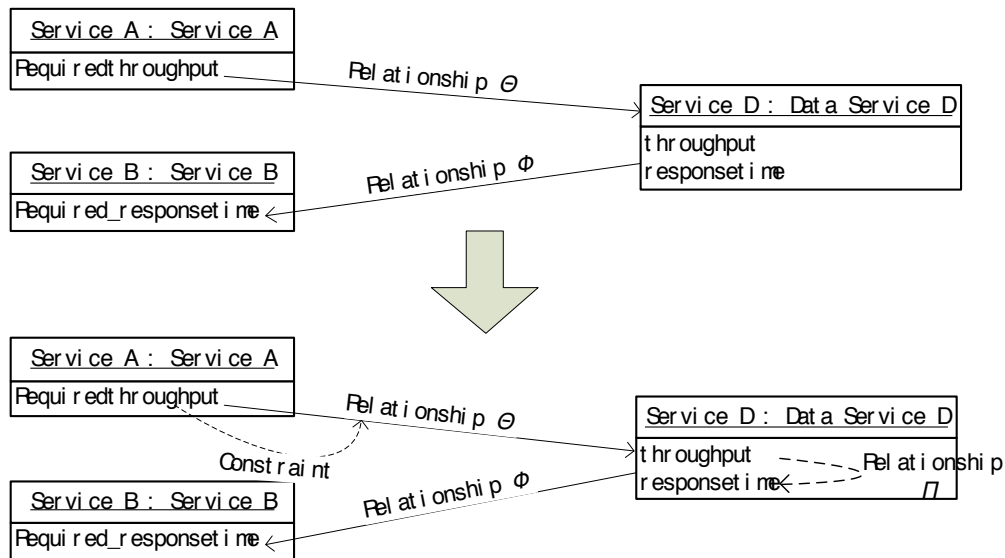


Figure 4.5 Relationships and Constraints

Relationships in an information domain are connections between entities, which are sets of attributes (as illustrated in Figure 4.5). Therefore, relationships are connections of attributes. In Figure 4.6, there is a relationship between attribute *requiredthroughput* and attribute *throughput*, which can be represented as:

$$\Theta = \text{Equal}(A.\text{requiredthroughput}, D.\text{throughput}) = \text{true}$$

Because this is also a constraint limiting this relationship, I have to consider this constraint during logical reasoning related to this relationship. The constraint of this relationship is:

$$(1000KB/s < A.requiredthroughput < 2000KB/s) \wedge (A \in Domain A) \wedge (D \in Domain A) \\ \wedge (A.requiredthroughput \in A, D.throughput \in D).$$

If I consider this constraint, the relationship becomes two sub-relationships: $\Theta'(c)$ and $\Theta - \Theta'(c)$ (as illustrated in Figure 4.6 b). $\Theta'(c)$ represents one sub-relationship that holds when constraint c is true or becomes effective. $\Theta - \Theta'(c)$ represents the other sub-relationship that is not affected by constraint c . However, in semantic extension, I consider these two sub-relationships as one complete relationship, which can be expressed as $\Theta = (\Theta - \Theta'(c)) \cup \Theta'(c)$. If constraints are not active at certain time or under certain conditions, the $\Theta'(c)$ sub-relationship will be empty and the relationship Θ will become $\Theta = (\Theta - \phi) \cup \phi = \Theta$.

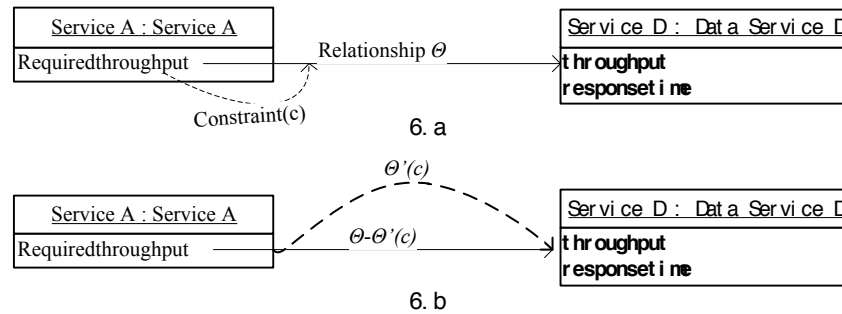


Figure 4.6 Relationships and Sub-relationships

In semantic extension, relationships include all temporal logic relationships, such as “Earlier than”; and other logical relationships, such as “Equal”, “Larger”, “Smaller”, and

“Negative”. The relationships in the knowledge-augmented temporal logic are inherited from the temporal logic and propositional logics.

Constraints on an action restrict attributes and the relationship of this action. In this paper, I use symbol Δ to denote this type of constraints. Again, relationship works on a Cartesian product of two entities, and can be expressed as $\Theta = \text{Equal}(\text{requiredthroughput}, \text{throughput})$, where *requiredthroughput* and *throughput* are attributes in entities. If there is a constraint c on a relationship, the constrained relationship becomes $\Theta'(\Delta) = \text{Equal}'(\text{requiredthroughput}, \text{throughput}, \Delta)$, where Δ is one or a set of constraints. In semantic extension, constraints are expressed as predicates that returns whether the constraints are satisfied or not. Only when constraints are satisfied, an attribute or relationship can change to a certain value.

A semantic extension abstracts certain information from an information domain. Now I can give a definition for semantic extension.

Definition 12: A semantic extension contains $\{\alpha\}, \{E\}, \{R\}, \{\Delta\}$, that are, correspondingly, the attributes set, entities set, relationships set and constraint sets from one information domain. $\Sigma = \{\{\alpha\}, \{E\}, \{R\}, \{\Delta\} \mid E \subseteq \alpha, E \neq \emptyset, R = E \times E\}$.

$\{\alpha\}, \{E\}, \{R\}, \{\Delta\}$ are attribute set, entity set, relationship set, constraint sets in a information domain. An entity is a sub set of $\{\alpha\}$, and entity cannot be empty. A relationship works on a Cartesian product of two entities.

Attributes in an information domain are associated not only with entities in the domain but also with attributes describing properties of the domain. These are domain attributes

that usually do not constitute entities. However, domain attributes may be added to entities under certain circumstances. For example, when two semantic extensions merge together, the attribute *domain ID* may become an attribute of an entity.

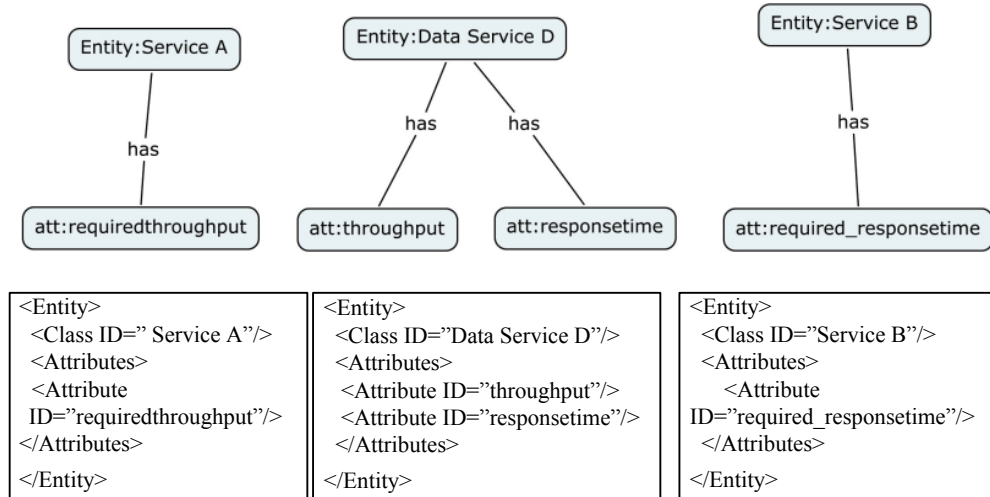


Figure 4.7 Example of an Entity in the Knowledge Base

The Figure 4.7 shows examples of entities in the knowledge base. The diagram shows each entity has its own attributes, which is expressed in XML format in the lower forms. In these examples, an entity includes a *class ID* and a set of *attributes*. A relationship includes an *ID* that is the name of this relationship. A relationship also includes two entities: one *Executor* and one *Target*. In an *Executor*, there is an attribute that involves in this relationship. In the relationship *Equal*, there is a constraint in an attribute of the *Target*. This constraint limits the maximum value of the attribute *throughput* of entity *Data Service D*. In another example, the relationship *Larger* has two entities: *Service B* and *Data Service D*. There is a constraint attached to the *Service B*, which limits the maximum of

required_responsetime of *Service B*. In original knowledge base, entities and attributes are well presented. The Semantic Extension focuses on relationships and constraints. Each relationship contains constraints in its attributes.

In the Figure 4.8, each relationship has two entities: *Executor* and *Target*. Each entity is consisted with a set of attributes. In this example, the diagram only shows attributes that involved in the relationship. There are two attributes in *Executor* and *Target* respectively in relationship *Equal*. The attribute *requiredthroughput* has a property on its value. This property is *Adjustable*, which indicates the *Executor* can change this attributes to any value that smaller than the constrained value 2MB/s. The attribute *throughput* in *Target* has a property that is *Variable*. This property indicates this attribute will be changed according to environment. The operator in this relationship is *Equal* or =. This operator is used to perform validation of whether this relationship is hold. This example presents if and only if $A.\text{requiredthroughput}$ is equal to $D.\text{throughput}$, and the $A.\text{requiredthroughput} < 2\text{MB/s}$, the relationship *Equal* is hold. The second example presents if and only if $B.\text{required_responsetime}$ is larger than $D.\text{responsetime}$ and $B.\text{required_responsetime}$ is set to 100ms, the relationship *Larger* is hold. In these two relationships, there are dynamic attributes, such as $D.\text{throughput}$ and $D.\text{responsetime}$. This information is stored in the semantic extension.

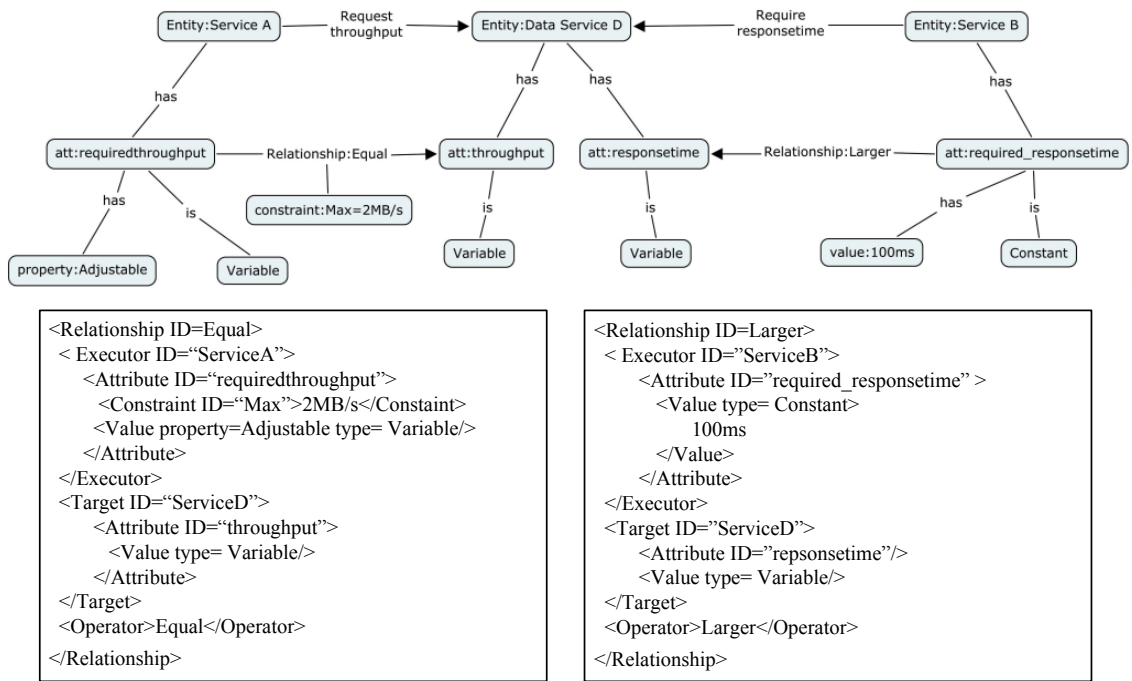


Figure 4.8 Examples of Relationships in the Semantic Extension

In Equal relationship, there is an arrow from Service A’s attribute *requiredthroughput* point to *Data Service D*’s attribute *throughput*. The origin attribute of this relationship is *requiredthroughput*, which is the superscript of the relationship in logic formulae. The relationship is the predicate of Cartesian product of two entities. To maintain a relationship between two entities is to maintain the truth value of the predicate. In the relationship *Equal*, the value of two attributes *requiredthroughput* and *throughput* from two different entities are the same. Because the attribute *requiredthroughput* is adjustable by users, when the value of *requiredthroughput* changes, the value of predicate *Equal* will not be hold as true. In order to maintain the truth value of predicate *Equal*, the value of another attribute *throughput* in *Data Service D* has to change. In this process, reason of truth value of predicate *Equal* changes is the change of attribute *requiredthroughput*. In a relationship,

the attribute that leads the change of truth value of predicate is the superscript of this relationship. The attribute that cover the change to maintain the truth value of the predicate is the subscript of this relationship. In Figure 4.8, the attribute *throughput* is the one that changes to maintain the truth value of the predicate *Equal*. The *Equal* relationship is expressed as: $Equal^{A.requiredthroughput}_{D.throughput}$. In *Larger* relationship, the attribute *responsetime* is dynamic and affected by the attribute *throughput* in the same entity. When *Data Service D* provides too much throughput for *Service A*, *Data Service D* may increase its response time to *Service B*. Therefore, the truth value of predicate of *Larger* relationship cannot be hold. In order to maintain this relationship, the attribute *required_responsetime* supposes to increase as well. However, this attribute is a constant that cannot change according to environment. Therefore, the attribute *required_responsetime* is still subscript of *Larger* relationship. The *Larger* relationship is expressed as $Larger^{B.required_responsetime}_{D.responsetime}$.

Domain ontology is a conceptualization for an information domain, which also provides a formal way to represent domain information. Information in semantic extension is extracted from one domain, so I use ontology to express and store domain information in a semantic extension. Usually I use one semantic extension to represent one information domain. One knowledge base can contain one or more semantic extensions, which depends upon the scope of this knowledge base. In a semantic extension, relationships are associated with corresponding attributes. When constraints on these relationships are satisfied, changes of individual attributes will affect relationships. Since certain constraints will change over time, I use temporal logic to represent these dynamic constraints.

For example (as illustrated in Figure 4.5), relationship Θ , which connects services A , B and D , has one explicit attribute *requiredthroughput* in service A and one implicit attribute *throughput* in *Data Service D*. And there is a constraint Δ on *requiredthroughput* and Θ . Relationship Φ connects one attribute *required_responsetime* in service B and one attribute *responsetime* in *Data Service D*. And there is a constraint Δ' on *required_responsetime* and *responsetime* in relationship Φ . Relationship Π connects attributes *throughput* and *responsetime* in *Data Service D*. When constraints are satisfied, *requiredthroughput* will affect Θ and *throughput* in *Data Service D*. And this change is transferred through relationship Π and will further affect *responsetime* in *Data Service D*. Then the change of *responsetime* will change the relationship Φ . This situation can be represented as the following rule:

$$\begin{aligned}
& \forall t, (T < t) \wedge \text{HoldsAt}(\Theta_{D.throughput}^{A.requiredthroughput} = \text{Equal}_{D.throughput}^{A.requiredthroughput}, t) \\
& \wedge \text{HoldsAt}(\Pi_{D.responsetime}^{D.throughput} = \text{Balance}_{D.responsetime}^{D.throughput}, t) \\
& \wedge \text{HoldsAt}(\Phi_{B.required_responsetime}^{D.responsetime} = \text{Larger}_{B.required_responsetime}^{D.responsetime}, t) \\
& \wedge \text{HoldsAt}(\Delta, T) \wedge \text{HoldsAt}(\Delta', T) \wedge \text{Change}(A.requiredthroughput, t) \\
& \Rightarrow \text{Change}(D.response, t) \wedge \text{Change}(D.throughput, t) \wedge \text{Change}(\Phi, t)
\end{aligned}$$

In this logical expression, constraints Δ and Δ' hold after time T . Therefore, after time T , if explicit attribute *requiredthroughput* in service A changes, implicit attribute *throughput* in *Data Service D* will change; attribute *response_time* in *Data Service D* will change because of relationship Π ; attribute *response_time* in *Data Service D* will further affect relationship Φ .

4.3 Rules in Conflict Analysis and Conflict Reconciliation:

In conflict analysis, I usually assume the executor of an action is an entity, and the target of an action is another entity. There are three major categories of conflicts: (1) conflict of duty, (2) conflict of interest, and (3) different executors perform different actions on a single target, and the outcome of each action is incongruent with each other. To represent these conflict types, let us consider the following elements in the general policy model. Executor S has an attribute χ , which is an explicit attribute for relationship Θ . There is also an implicit attribute i in executor S . Attribute i is an explicit attribute for relationship Φ . Target O has an attribute v and an attribute j . Relationship Θ connects attribute χ and v . Relationship Φ connects attribute i and j . In the following conflict analysis rules, relationship K and A indicate two actions respectively. There are two constraints δ and ω restricting Θ and Φ . K and A are two actions between S and O . Relationship Θ belongs to action K , and Φ belongs to action A .

Conflict of duty: K and A are two actions between S and O . They contain relationship Θ and Φ respectively. If their explicit attributes change, these two actions cannot be performed at the same time:

$$\begin{aligned}
& \exists t, t' \mid HoldsAt(\delta, t) \wedge HoldsAt(\omega, t) \wedge K(S, O) \\
& \wedge \Lambda(S, O) \wedge (t < t') \wedge HoldsAt(\Theta_i^x, t') \wedge HoldsAt(\Phi^i, t) \\
& \Rightarrow change(\chi, t') \wedge change(i, t') \\
& \Rightarrow conflictofDuty(K(S, O), \Lambda(S, O), t')
\end{aligned}$$

When action K is performed, attribute χ changes, then the implicit attribute i will also change. However, action A is performed at the same time, so attribute i has to change too.

These two changes cannot happen at the same point in time. Thus a conflict of duty happens. When two actions cause a conflict, there is one action has higher priority, such as action $K(S,O)$. The system should allow $K(S,O)$ instead of action $\Lambda(S,O)$. Reconciliation rule for this type of conflict is shown as follow:

$$\begin{aligned} & HoldsAt(conflictofDuty(K(S, O), \Lambda(S, O), t'), t) \\ & \wedge HighPriority(K(S, O)) \\ \Rightarrow & Trajectory(permit(permit(S, O, K(S, O)), deny(S, O, \Lambda(S, O))), t) \end{aligned}$$

Conflict of interest: K is an action between S and O , and Λ is an action between S and O' . There is a relationship Θ between S and O . There is another relationship Φ between S and O' . The constraint δ constrains the attribute i in entity O , while attribute j in entity O' cannot change at the same time.

$$\begin{aligned} \exists t, t' \mid & HoldsAt(\delta, t) \wedge HoldsAt(\omega, t) \wedge K(S, O) \wedge \Lambda(S, O') \\ & \wedge (t < t') \wedge HoldsAt(\Theta_i^x, t') \wedge HoldsAt(\Phi_j^v, t) \\ \Rightarrow & HoldsAt(\Theta_i^x, t') \wedge HoldsAt(\Phi_j^v, t) \wedge Change(\chi, t') \wedge Change(v, t') \\ \Rightarrow & Change(i, t') \wedge Change(j, t') \\ \Rightarrow & conflictofInterest(K(S, O), \Lambda(S, O'), t') \end{aligned}$$

When action K is performed, attribute χ will change, and the implicit attribute i will also change. At the same time, if action Λ is performed, attribute v and attribute j both will change. There will be a conflict according to the constraint δ . If one of these actions has higher priority than another action, this action will be granted and another action will be denied but the system. This reconciliation rule is shown as follow:

$$\begin{aligned}
& \text{HoldsAt}(\text{conflictOfInterest}(K(S, O), \Lambda(S, O'), t'), t) \\
& \wedge \text{HighPriority}(K(S, O)) \\
\Rightarrow & \text{Trajectory}(\text{permit}(\text{permit}(S, O, K(S, O)), \text{deny}(S, O, \Lambda(S, O'))), t)
\end{aligned}$$

Different executors perform different actions on a single target, and the outcome of each action is incongruent with each other:

K is an action between S and O, and A is an action between S' and O. There is a relationship Θ between S and O. There is another relationship Φ between S' and O. In this case, relationship Θ connects attribute χ in entity S and attribute i in entity O; relationship Φ connects attribute v in entity S' and attribute i in entity O. When attribute χ and v change, these two actions cannot be performed at the same time.

$$\begin{aligned}
& \exists t, t' \mid \text{HoldsAt}(\delta, t) \wedge \text{HoldsAt}(\omega, t) \\
& \wedge K(S, O) \wedge \Lambda(S', O) \wedge (t < t') \wedge \text{HoldsAt}(\Theta_i^x, t') \wedge \text{HoldsAt}(\Phi_j^v, t) \\
\Rightarrow & \text{HoldsAt}(\Theta_i^x, t') \wedge \text{HoldsAt}(\Phi_j^v, t) \\
& \wedge \text{Change}(\chi, t') \wedge \text{Change}(v, t') \\
\Rightarrow & \text{Change}(i, t') \wedge \text{Change}'(i, t') \\
\Rightarrow & \text{conflictOfDiff}(K(S, O), \Lambda(S', O), t')
\end{aligned}$$

When action K is performed, attribute χ will change, and implicit attribute i will also change. At the same time, if action A is performed, attribute v and attribute i in entity O will also change. Thus, this type of conflict occurs. If these is an action has higher priority, this action is permitted by the system, and the low priority action will be denied. The reconciliation rule is shown as follow:

$$\begin{aligned}
& \text{HoldsAt}(\text{conflictOfDiff}(K(S, 0), \Lambda(S', 0), t'), t) \\
& \wedge \text{HighPriority}(\Lambda(S', 0)) \\
& \Rightarrow \text{Trajectory}(\text{permit}(\text{permit}(S, 0, \Lambda(S', 0)), \\
& \text{deny}(S, 0, K(S, 0))), t')
\end{aligned}$$

4.4 Experiment of Policy Conflict Analysis Component

In the web services case, I have two types of policies (access control policies and quality of service policies). Both are included in our experiment policy sets. For the experiment, two policy sets are established. One policy set only contains static conflicts (the static policy set). Another policy set contains conflicts that are caused by dynamic attributes and dynamic relationships (the dynamic policy set). In the static policy set, required information for analysis is static and is contained in the policies themselves. So during the analysis process, temporal logic does not need additional information since static conflicts are not caused by dynamic attributes or relationships. Each policy set contains 100 policy segment pairs. In these 100 pairs, there are 45 and 57 conflicts in two policy sets respectively. I also implement three other conflict analysis algorithms and compare their results with those generate from our knowledge-augmented temporal logic approach, as well as with the result from a human domain expert's manual investigation. In Table 4.1, under column "Static Policy Set" and "Dynamic Policy Set", there are three elements in each row. In each three-element tuple, the first element is the number of conflicts found by that approach; the second element is the total number of conflicts in the policy set, and the third number is the total policy segment pairs in the policy set.

	Static Policy Set		
	Detected Conflict	Policy Conflict	Policy Segment
Event-driven model[58]	45	45	100
IPCDR [59]	45	45	100
Temporal Logic	45	45	100
Result of TL with Semantic Extension	45	45	100
Human	43	45	100
	Dynamic Policy Set		
	Detected Conflict	Policy Conflict	Policy Segment
Event-driven model[58]	30	57	100
IPCDR [59]	30	57	100
Temporal Logic	30	57	100
Result of TL with Semantic Extension	55	57	100
Human	35	57	100

Table 4.1 Comparison of Different Policy Conflict Analysis Algorithms (A)

In this experiment, Event-driven model [58] and IPCDR [59] focus on the policies themselves without policy domain information. These algorithms can analyze policies with explicit attributes only. If there are changes in a relationship and implicit attributes, these algorithms cannot detect conflicts accurately, because they do not consider domain information during its analysis process. In the experiment on dynamic policy set, results show that temporal logic integrated with semantic extension has a much better accuracy than that of pure temporal logic. In the analysis process, semantic extension provides dynamic attribute and relationship information to supplement temporal logic rules. This extra information helps our system identify entities and build constraints on different attributes and relationships. For the dynamic policy set, temporal logic finds 30 policy conflicts caused by dynamic attributes. Actually, more conflicts rooted 27 from dynamic relationships and dynamic constraints on relationships. These conflicts are detected by temporal logic with semantic extension. At the end of the experiment, I also ask a system

administrator who has the knowledge of web services and web service policy to analyze these two policy sets manually. As shown in Table 4.1, most conflicts in static policy sets are identified by this human expert. However, for the dynamic policy set, only 35 conflicts are found by this human expert. Although a human expert can incorporate environment information into policy analysis, the complexity of the environment information is still the major obstacle in manual analysis. The result of manual analysis heavily depends upon the analyzer's experience. Therefore, I clearly see that without a proper knowledge base, dynamic conflicts are hard to detect. Semantic extension can provide such information for logical analysis, and the analysis engine can handle very complex situation. The entire framework can provide recommendation for subsequent reconciliation with a proper knowledge base.

In the sensor system case, I choose three sets of policies (A, B, C). The policies in these sets come from two different sensor systems. I collect these policies before these two sensor systems start collaboration. 30 policies are from one sensor system, and the other 30 are from the other. There are 20 policy pairs in each set. There are 15 static conflicts in set A, 16 dynamic conflicts in set B, and 13 dynamic conflicts in set C. Conflicts in policy set B are dynamic conflicts, but there is no hidden relationships, explicit attribute or implicit attribute involved in any conflict. Conflicts in policy set C are also dynamic conflicts, but certain hidden relationships, explicit and implicit attributes are involved. The analysis result is shown in Table 4.2. In this table, under column "Policy Set A", "Policy Set B" and "Policy Set C", there are three elements in each row as well. The numbers in each row have the same meaning as in Table 4.1.

	Policy Set A			Policy Set B			Policy Set C		
	Detected Conflict	Policy Conflict	Policy Segment	Detected Conflict	Policy Conflict	Policy Segment	Detected Conflict	Policy Conflict	Policy Segment
Event-driven model[58]	15	15	20	16	16	20	10	13	20
IPCDR[59]	15	15	20	15	16	20	8	13	20
Result of TL	15	15	20	16	16	20	8	13	20
Result of TL with Semantic Extension	15	15	20	16	16	20	13	13	20

Table 4.2 Comparison of Different Policy Conflict Analysis Algorithms (B)

In this experiment, three policy sets contain different levels of conflict. In set A and B, all algorithms provide similar results. However, in the Policy Set C, IPCDR and Temporal logic can only find 8 out of 13 conflicts. These 8 conflicts have either entity or action overlaps in their corresponding policies. If there are changes of relationships connecting entity or relationship attributes with environment attributes, the lack of knowledge support will result in a low accuracy of analysis result.

CHAPTER 5 NON-INTRUSIVE LOG PROCESSING

5.1 Service Event Pattern Learning

5.1.1 Event Pattern Learning Technique Survey

In a cloud computing environment, services are deployed in number of virtual machines. Service events are collected from all involved virtual machines. Service events contain service status information, service run-time variables, virtual machine status information and other service related data. Most services provide functions to collect service events and create service event logs. Through these service event logs, I can learn the behavior of services and their environment. Pattern learning techniques can help analyze service event logs to find service event patterns. Each service event pattern indicates one possible path that service state changes from a normal state to a critical state. Service events can be categorized into different event levels, such as hardware events, system level events, software events, maintenance events, etc. In order to detect critical event pattern and predict critical events, pattern learning and prediction algorithms are must in the service event monitoring component.

Sequential event pattern learning has been studied for decades [65, 66, 67, 68, 69 and 73], which is a good candidate mechanism for cloud system event detection and prediction. In [57], Agrawal discusses an Apriori-based pattern learning method called Generalized Sequential Pattern algorithm (GSP). This GSP algorithm screens all length-1 candidates in a database. Those sequences with a support that is less than the minimum support are filtered out. Then the algorithm screens the database k times to collect support count for

each length- k candidate and generates length- $(k+1)$ candidate sequences from length- k frequent sequences using Apriori. GSP algorithm generates a huge set of candidate sequences in multiple database scans, which is inefficient for large databases. In [74], authors use a level-wise association-rule algorithm to exploit anti-monotone and monotone constraints so that the problem's level-wise dimensions can be reduced. Each transaction, before being added to the support count, is reduced as much as possible, and only if it survives this phase, it will be used to count towards the support for candidate item sets. Each transaction, which can arrive this counting phase at iteration k , is then reduced again as much as possible; and only if it survives this second set of reduction, it will be written to the transaction database for the next iteration. Artificial neural network is another family of learning and prediction mechanisms that utilizes neuron functions to provide outputs based on a large number of inputs. However, artificial neural network alone may be difficult to satisfy the close to real-time requirement for cloud system reliability management. So certain adjustment and improvement together with other auxiliary mechanisms are necessary. For event prediction, two principal approaches to critical event prediction based on previous occurrences of failure can be determined. One is estimation of the probability distribution of a random variable for the time to next failure. The other type of approach builds its estimation based on the co-occurrence of two or more critical events [75].

Furthermore, authors of [77] suggest that the behavior of individual services in a service process must be monitored in order to settle any responsibility issue and to meet the overall quality requirements from its consumers. To provide a solution, they suggest

any under-performed service should be replaced immediately to ensure the required level of service quality. It is true that quality requirements and under-performed services are the imperatives for SOA. However, without a clear definition of quality requirements for services, the motivation of this solution seems obscure. Also, “by using which standards a recovery application can be invoked to start healing mechanism” is still a question in these two papers. Similar issues also arise in cloud computing systems. For instance, in the RESERVOIR architecture [76], a service manager is responsible for monitoring the deployed services and adjusting their capacity. At the same time, the service manager also needs to take care of the number of Visual Execution Environment (VEE) instances as well as their resource allocation to ensure SLA compliance and alignment with high-level business goals. Although this article mentions adjusting service capability and resource allocation, there is not a set of explicit requirements of quality defined for services, especially for a multi-tenant environment. For another instance, Roy Campbell, etc. [75] propose a cloud computing test-bed to create unified and coherent resources, rather than several completely separated clusters that provide reliable functionalities. Although, in their Open Cirrus service stack, the lowest level service is based on a notion of physical resource set (PRS), there is still no mechanism to provide any monitoring and recording functionalities for tracking the existence of system resources. In [69], authors present a BIDE (BI-Directional Extension) algorithm that is adopted to learn event patterns from service event stream. This BIDE algorithm mines frequent close patterns in a given event dataset and prunes irrelevant pattern branches more quickly so that the search space becomes deeper by using the BackScan pruning method.

5.1.2 Hash Table with Reversed Frequent Pattern Tree

One most important feature or advantage of Hash table is the search speed, which is a constant time. A good hash function can greatly reduce search time in hash table search. In previous section, I summarize the BIDE algorithm which mines the event sequence database and find out all frequent closed patterns. It consists of Forward-checking, Backward-checking and BackScan pruning methods. This algorithm constructs a complete set of frequent closed pattern in an event sequence database, and a frequent closed pattern tree. During the real-time pattern learning process, the frequent pattern tree will change, when new pattern are found. And new frequent patterns are not closed, so the frequent pattern tree is not a closed frequent pattern tree anymore. However, during the learning process, I still try to compress the pattern tree to reduce the memory consumption. When the hash occupancy rate reaches a certain number, the rehash function is triggered. The rehash function hashes all existed node into a new larger hash table. The hash occupancy rate is set according to previous experience.

In the traditional tree structure, parent nodes are connected with children nodes through links. However, in our approach, there is only one link from one parent node to its first child node. This child node is also connected with other children node through sibling links. The Figure 5.1 shows a structure of a node. Child node pointer points to its first child node; Sibling Node pointer points to its next sibling node; Parent node pointer points to its parent node; Next node pointer points to next node in the linked list. In following presentation, figures only show links that related to their topics.

Pattern	Child Node pointer	Sibling Node Pointer	Parent Node Pointer	Next Node Pointer
----------------	--------------------	----------------------	---------------------	-------------------

Figure 5.1 Structure of a Node

Because children nodes contain a pointer that points to their parent nodes, I call it a reversed tree. The Figure 5.2 illustrates an example of a reversed pattern tree. In this figure, only parent node links are shown. Although there is no directed link from parent nodes to every child node, because of nature of pattern stream, from node “A” to node “AB” I can simply grow the node “A” with an event “B”. If there is an “AB” node exists in the tree, I can increase its statistic information; otherwise, I can create a new node with its parent node pointer points to node “A”.

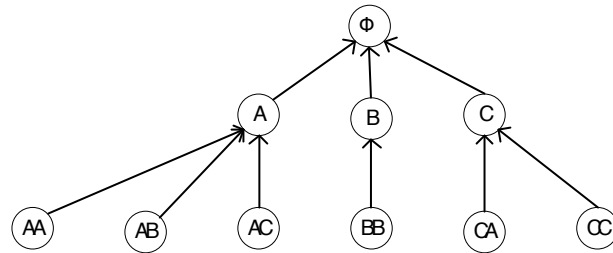


Figure 5.2 Reversed Frequent Pattern Tree

There is another problem that I cannot find a path from the root to any leaf node in the reversed pattern tree. In order to solve this problem, each node is stored in a linked list node, and there is a hash table that stores indexes and pointers that point to every node in reversed pattern tree. An entry in this hash table contains an index that calculated by a hash function, and a pointer points to a linked list node. A linked list node contains event pattern name and statistic information like support. A linked list node also has two pointers. One

pointer point to next linked list; and another pointer point to its parent node in the reversed pattern tree. So a linked list node can be presented as {pattern, next node pointer, parent node pointer}. The Figure 5.3 illustrates a reversed pattern tree using linked list node.

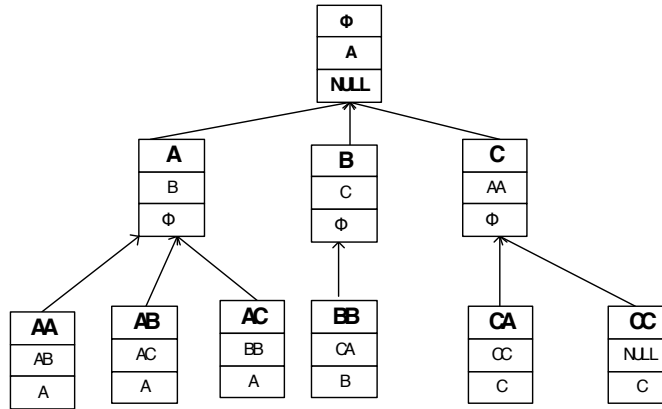


Figure 5.3 Reversed Pattern Tree using linked list node

These linked list nodes also connected through their first pointer. Then I have a linked list L (shown in Figure 5.4). This linked list contains all patterns in the system. When a new pattern appears, the system calculates an index for this pattern, creates a linked list node and sets the pointer in the corresponding hash table entry to the new linked list node. Then the new linked list node is added to the end of the linked list L. If there is a collision, the new linked list node will be inserted into the linked list L at the position after nodes that have the same index. The collision linked list is also a part of the linked list L, for example, the nodes “AA”, “AB” and “AC” are in the same collision linked list. The max length is also a parameter to trigger the rehash function.

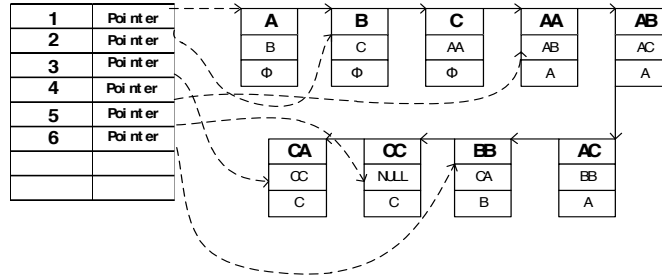


Figure 5.4 Example of Hash Table and Linked List

The Figure 5.4 shows a visualized example of Hash table T and linked list L. If the pattern “AC” is a new frequent pattern, the algorithm calculates the index for “AC”. In this example, I assume the pattern “AA”, “AB” and “AC” have the same index value, and “AA” and “AB” already exist in the linked list L. The algorithm finds the linked list node “AA” according to the hash table index “4”, and checks the index of linked list node “AB”, which is connected with node “AA”. The index of node “AB” is equal to the index of new pattern “AC”. Then the algorithm checks the index of node “BB”, which is connected with node “AB”. Because the index of node “BB” is not equal to the index of new pattern, the algorithm inserts the new pattern “AC” between linked list node “AB” and node “BB” as shown in the Figure 5.4.

In order to increase the search speed of finding a leaf node for a given node, each node has two more pointers that link parent node and children nodes. A node in the reversed frequent pattern tree has a child pointer, which points to one of its children nodes. This pointer is used to travel from parent node to children nodes (dotted line). Another pointer in a node is sibling pointer, which points to next sibling node. The child pointer and sibling

build a path from parent node to all children nodes (dash-dot line). The Figure 5.5 shows a reversed frequent pattern tree with child pointers and sibling pointers.

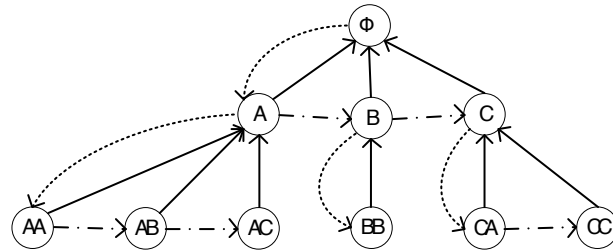


Figure 5.5 Reversed Pattern Tree with Child Pointers and Sibling Pointers

The Figure 5.6 illustrates the algorithm to insert a new frequent pattern into the Hash Table T and the linked list L. The algorithm first exams whether there is a hit in the hash table according to the calculated index of new pattern. If the corresponding table entry is null, then the algorithm checks whether the new pattern meets the minimum support thresholds. If the new pattern fulfills the requirement of minimum support thresholds, create a new linked list node N and add this linked list node to the end of the linked list L. Then the algorithm set the parent pointer of the new node to its parent node. If it is not null in the corresponding hash table entry, the algorithm exam whether there is a node for the input pattern already exists. If the input pattern is a new frequent patter, create a new linked list node, insert the node to the linked list L, and return the index of new frequent pattern.

Any event pattern should have one parent node. The parent node contains a pattern that is a sub-string of current pattern. The length of parent pattern is not necessary to be current pattern length-1. The length of parent pattern can be shorter, because some patterns may be merged with its children during the compress process.

```

New Frequent Pattern Insertion Algorithm
FrequentPattern_Insertion(HashTable T, Linked_List L, Event Pattern P, min_sup)
Input:  a Hash Table T;
a Linked_List L that stores event pattern P, a minimum support threshold min_sup)
Output: index of Event pattern P in Hash Table T
1: index=Hashi(P);
2: if T[index] is empty
3: {   if P.support > min_sup;
4:     {
5:         create a linked list node N;
6:         store P in N;
7:         add N to the end of Linked List L;
8:         call Parent_Index(N);
9:     }
10:  else
11:  return null;}
12:else
13:{
14:  if Pattern P is already in L
15:    update the statistic of exist node;
16:  else
17:    create a linked list node N;
18:    get the linked list node M from the hash table entry pointer;
19:    create a temporary node Temp=M.next;
19:    if linked list node Temp is not null:
20:      Insert N before the node Temp;
22:    call Parent_Index(N);
23:    return index;}

```

Figure 5.6 New Frequent Pattern Insertion Algorithm

Along with the online running of the system, the number of nodes in the linked list will become bigger and bigger, in order to save space and increase search speed, I have to compress the reversed pattern tree. In order to compress the reversed pattern tree in a real-time, each leaf node checks its support and its parent node's support. If its support equal to its parent node's support, I can merge the leaf node and its parent node. If there are different supports, I do not merge leaf and its parent nodes. Because the real-time online learning and recognition have a very sensitive time requirement, the compress process only take place when there are enough CPU idle time. The Figure 5.7 illustrates the Reversed

Frequent Pattern Tree compress algorithm. The algorithm first checks whether the current has a parent node. If there is a parent node and the support of parent node is equal to support of current node, current node will set its parent node pointer to its parent's parent. And remove current node's parent node. This process is called recursively.

If a node in an event sequence tree has the same support with its parent node, this node is the only child node of its parent node. In order to prove this statement, I assume event sequence S is the parent node of event sequence S' . The event sequence S' can be expressed as $S+Q$. Q is the remainder event sequence in S' . Every time S' appears, the $S+Q$ appears. When support of S' increase, the support of S will also increase. If there is another child node of S appears, the support of S will also increase by this child node. Therefore, if the support of S is equal to the support of S' , the event sequence S' is the only child node of event sequence S .

The collision chain in the hash table is related with the hash function. In the experiment section, I use FNV algorithm to calculate the index for each event pattern string. FNV is a non-cryptographic hash algorithm. The length of collision chain is a criterion for rehashing function. Another criterion is the hash occupancy ratio. When one of these criteria meets the threshold, the system is ready to start rehash function. The rehash function will not immediately, but it will start when the system has some ideal CPU cycle.

```

Compress_Tree(Linked_List L)
Input: a linked list L that contains all frequent patterns;
Output: a compressed Linked list;
1: linked_list_node N=L.firstnode;
2: loop:
3:   CheckParentNode(N);
4:   if Node.next!=NULL;
5:     Node=Node.next;
6:   else
7:     return;

CheckParentNode(Linked_List_Node N)
Input: a linked list node N;
Output: NULL;
8: if N.parent is not NULL
9:   if (N.support == N.parent.support)
10:    Compress N and N.parent node
11:    Remove(N)
12:    CheckParentNode(N);

Remove (Linked_List_Node N)
Input: a linked list node N;
Output: null;
13: If N.Child!=NULL
14:   Pointer Temp=N.child;
15:   Loop:
16:    Move N's children nodes to child node list of N's Parent node.
17: If N.Parent.Child!=N
18:   Pointer Temp= N.Parent.Child;
19:   Loop:
20:    if Temp.Sibling!=N&Temp.Sibling!=Null
21:      Temp=Temp.Sibling;
22:    else
23:      Temp.Sibling=N.Sibling;
24: Remove node N from the N.next list;

```

Figure 5.7 Reversed Frequent Pattern Tree Compress Algorithm

5.2 Event Pattern Detection

System event pattern recognition is the second step of critical event prediction. In this step, I have to read an event sequences, and search frequent pattern database that try to find out a matched frequent pattern. If there is a matched frequent pattern, I also have to update the statistical data for further recognition and prediction.

In a hash table, to search an existed item, I can use hash function to directly calculate the index of this item. If the hash table entry with this index contains this item, it will report a match. If there is not a match in this step, I have to consider two situations. One situation is that the hash table entry of this index is empty, which means there is no such item can match given data item. Another situation is that there is a linked list in this hash table entry. If the given pattern can match any item in this linked list, I also can report a match. If there is no match in the linked list, I will report an unmatched notification.

If the given event sequence can be found in the hash table, which means this event sequence is a frequent closed pattern. And this item can be found in the event sequence tree. I will update the statistical data of this event pattern. The Figure 5.8 shows an example of a process of recognition. In this example I have an event sequence “AC”, the index of “AC” is 4. The pointer of index 4 is pointing to pattern “AA”, which is not the target pattern. Then I will try the next node in the linked list. The next node is “AB”, which has the same index. Because the “AB” is not the target pattern, I will continue to next node that contains pattern “AC”. The pattern “AC” is the target pattern, and it will be return as a hit. If the target pattern does not exist in the pattern list, a NULL will be return. For example, if the index of an event sequence “AD” is 4, which will lead to the pattern “AA”. After I traverse

5.3 Event Pattern Prediction Algorithm

5.3.1 Filtered-Multi Dimensions Neural Network

The information in service event log represents running status of a service. In event sequences, each event contains a set of system attributes. Events can be categorized into number of event types or event levels according to attribute types and value of attributes within events. Events in the same type or level usually contain the same or similar set of attributes. Attribute set of an event sequence is the union set of attribute sets of events. Sequences of service events are transmission processes of this union attribute set. In the meanwhile, attributes in an event have different severities. Some attributes represent event types, and some attributes represent the location of events. These attributes can be organized by their severities. The most important attribute is the primary attribute for an event. In pattern prediction process, attributes with higher severity will be used first to predict future events. I create several attribute trees to help build prediction model and then predict possible event. Each attribute has one corresponding tree. Event patterns can be identified by their primary attributes represented by a sequence. For example, an event sequence ABC, which A's attribute set is $\{a1, a2, a3\}$, B's attribute set is $\{b1, b2, b3\}$, and C's attribute set is $\{c1, c2, c3\}$. Therefore, this event sequence can be presented as $\langle a1, b1, c1 \rangle$. I call this attribute sequence primary attribute sequence. In a primary attribute sequence, attributes are primary attributes of events. Figure 3 shows an attribute tree with severity. In this attribute tree, only the primary attributes have been considered. According to severity, I can build attribute forest to present an event sequences, and these trees can be searched or processed parallel.

The neural network algorithms can handle time series events and forecast future events. The basis function neural networks are a class of neural networks. The base function neural networks produce weighted sum of a number of base functions. A wavelet neural network (WNN) [79] is an alternative to the classical feed-forward neural network (FFNN) [80] for approximating arbitrary nonlinear functions, inspired by both the FFNN and wavelet theory. The WNN has been successfully applied into the function learning and time series predictions. Wavelet functions include continuous wavelet transform and discrete wavelet transform. Since wavelets have shown their excellent performance in nonstationary signal analysis and nonlinear function modeling. In the simplest form of wavelet neural network, the input and output both have only one variable. The output of this simplest wavelet neural network can be defined as:

$$\psi_{\lambda,t}(u) = \psi\left(\frac{u-t}{\lambda}\right)$$

λ is the dilation parameter, and t is the translation parameter. And the structure of this simple wavelet neural network is shown in Figure 5.9.

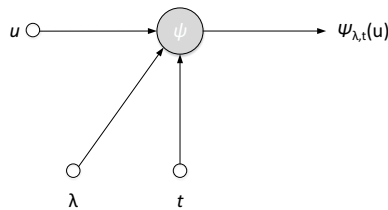


Figure 5.9 A Simple Wavelet Neural Network

The n -dimensional wavelet basis function can be calculated by the tensor product of 1-D wavelets. Therefore, the output of all hidden layer neurons (wavelons) will be the same and can be written as

$$\psi_{\lambda,t}(u) = \prod \psi\left(\frac{u-t}{\lambda}\right)$$

Attributes of events cannot easily fit in this simple wavelet neural network. Each event is a multiple dimensional vector. Therefore, I need to employ a multidimensional wavelet neural network. In the multidimensional wavelet neural network, input event sequence of the model is considered as a sequence of multidimensional vectors. Figure 5.10 illustrates an architecture of an attribute-based multidimensional wavelet neural network. In the input layer, each input node read an event from event sequence, and pass event attribute set into next layer. The filter layer selects attributes base on the severity of event attribute set into next layer. The filter layer selects attributes base on the severity of attributes in an event. The severity information is provided by the knowledge-base of the prediction framework. The goal of this layer is to reduce the dimension of the input of neural network. The high number of dimensions makes the neural network cannot convergent and high time complexity.

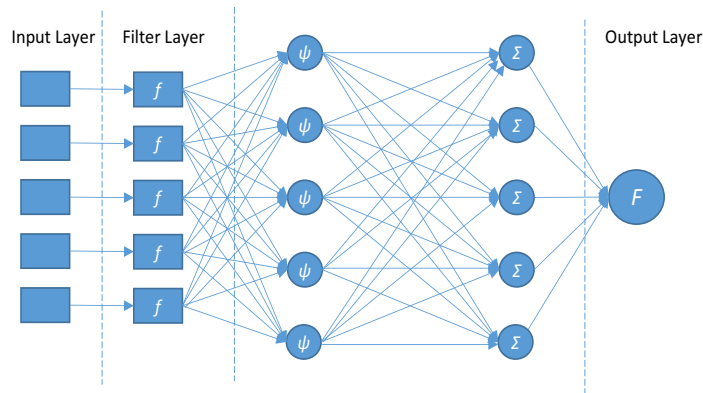


Figure 5.10 Layers of Filtered MD Wavelet Neural Network

Each type of system attribute has its own neuron to calculate the influence toward the system. Therefore, the second layer of the network extract every attribute and send them

into different neuron. The number of filter nodes is predefined according to the maximum number of event patterns. The output of filter layer is a set of related attributes of each event. This attribute set is defined in the log recording and pattern learning process. In each filter node, there is an attribute selector. Attribute selectors are designed for filter layer to choose primary attribute set and relation attribute from input event sequences. During the network learning process, the selector searches the knowledge base for each input event and provides most important attributes. Attribute selectors is also a control model to control the number of input dimension of the network. The control logic within attribute selectors is predefined and expressed as logical expression. For example, Fan speed is an input attribute, which is an integer value, with a normal range from 1000 to 5000. If the value of this attribute is out of its range, it indicates either there is some problem with the fan, or the system is overheated. When the fan speed is in the normal range, this attribute may not be considered as an input to the network. If the fan speed is out of normal range, the fan speed is an indicator of some abnormal situations. The logical expression to evaluate this attribute is:

$$E(t) = \begin{cases} false & \text{if } (t > 1000 \wedge t < 5000) \\ true & \text{if } (t < 1000 \vee t > 5000) \end{cases}$$

This logical expression will determine whether the fan speed attribute will be considered in the hidden layer. The input of fan speed is transferred into an indicator that affects the result of prediction. Service event sequences are time series data, therefore, the temporal logical expression can handle time-related attributes. For example, the fan speed is a time-related attribute in an event sequence. This attribute appears in multiple events and changes its value in the event sequence. Then I can use temporal logic to express this

attribute in different events. Neurons in the first layer calculate the output for next layer according to the output of filter layer. The last layer of nodes in the hidden layer is the summation layer. Each node in this layer get the sum of previous layer outputs

In virtualized environment, events come from different virtual machines, and these virtual machines consist of different information domains. Therefore, event attributes consist of cross-system information. Some attributes contain local node information; some attributes contain remote user information; and some attribute contains relationship information of current event and future events. This information can help us evaluate which attributes should be considered in the learning and prediction process. Events in the same pattern has stronger mutual relationship than other events. This relationship is mainly reflected by certain attributes. I call this type of attributes relation attributes. The number of relation attributes in an event pattern shows the level of interdependency between two events. The more relation attributes, the stronger relationship between the events in an event pattern. Taking relation attributes into the hidden layer of the WNN is one task of filter layer. In order to determine whether an attribute is a relation attribute in an event pattern, I use logical rules to evaluate event pattern attribute sets and store the relationship attributes in the knowledge base. In the learning and predicting process, the relationship attribute is another aspect that to calculate the final output. The temporal logic expressions and temporal rules are used in the neural network to filter input of each neuron according to time attribute of each events.

Because the input of the wavelet neural network in proposed framework is multidimensional, the output of the multidimensional wavelet neural network can be express as:

$$f(x) = \sum_{i=1}^M \omega_i \Psi_i(x) = \sum_{i=1}^M \omega_i |a_i|^{-\frac{1}{2}} \psi\left(\frac{x-b_i}{a_i}\right)$$

a_i is the scale parameter, and b_i is the translation parameter. Ψ_i is the wavelet activation function.

The learning process of this neural network establishes the connection of different attributes and events. In the learning set, the back propagation method is used in the training step. In order to increase the chance of convergence, I setup certain adaptive stopping criterion. Since the function computed by this WNN model is differentiable with respect to all mentioned unknown parameters, a standard back-propagation (BP) gradient descent training algorithm can be used with guided attribute selectors.

5.4 Experiments

In our experiment, I collect events from over 20000 event records from a cloud data center. The data center consists of a number of high performance computing systems. There are various services running in this data center. In order to test the learning and prediction, I divide the testing data set into two parts. One part is used for offline learning, the other part is used for online prediction.

Events in this dataset is recorded in time order. Each event contains its application id and node id attribute. The application id is used to differential services. One service usually

consists of number of nodes. These two attributes indicates the location of events. I divided the dataset into two parts, one is for offline learning and another one is for testing. In the first half of dataset, there are 8000 event records are used in offline learning, and 2000 event records are used for tuning the network. The second half are used for test of prediction.

In the experiments, I setup different candidates from the prediction results, and use these candidates to verify prediction results. If candidate events happen right after the prediction, I consider this prediction is correct. If candidate events do not appear after the prediction, I consider this prediction is incorrect. The result shows the number of candidates directly affects the accuracy of the prediction. However, larger number of candidates will affect the efficiency of prediction. In Figure 5.11, I illustrates the comparison of accuracy from wavelet neural network based algorithm and from an average one dependency estimator (AODE) based algorithm. The result also shows that the pattern library is not stable at the beginning. The pattern library is updated during event processes. New patterns are added into the pattern library, and patterns that cannot meet the minimum support are removed from the pattern library. The result demonstrates that when pattern library becomes stable, relation attributes and structural attributes can help wavelet neural network to generate results that are more accurate.

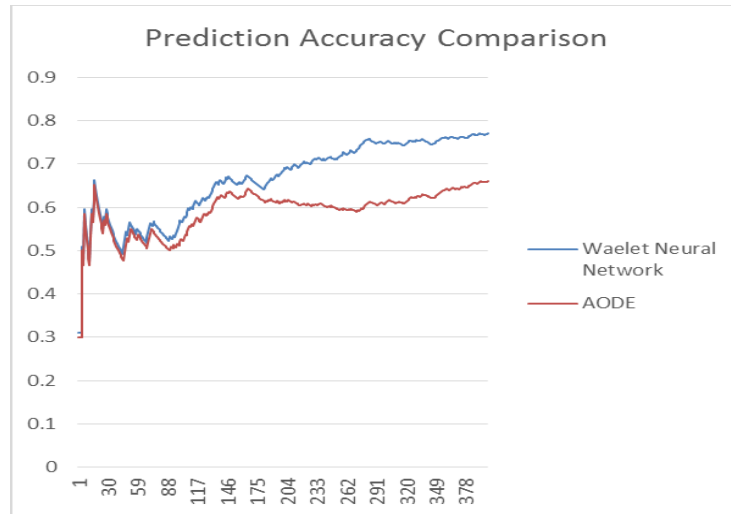


Figure 5.11 Prediction Accuracy Comparison of WNN and AODE

The pattern library is updated during the experiment. Figure 5.12 shows the performance of WNN, multi-dimension WNN and Filter multi-dimension WNN with different number of hidden nodes. This result shows the multi-dimension has better prediction result comparing to WNN algorithm. The filtering layer reduce the complexity of input dimension without reducing the accuracy. The severity of attributes can help the proposed framework to achieve better performance with high efficiency.

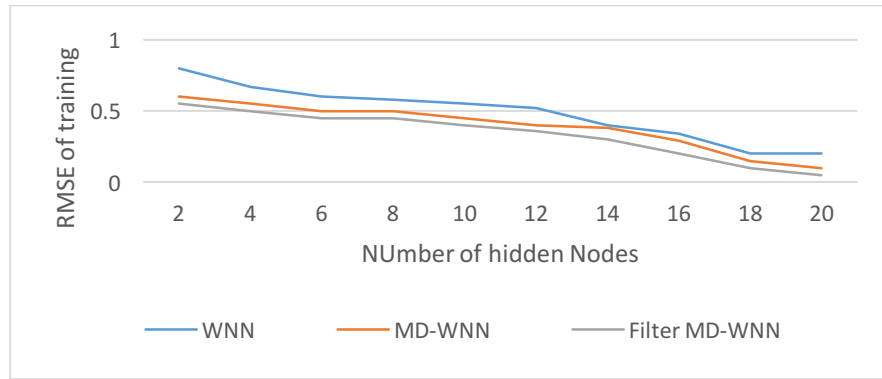


Figure 5.12 The performance of WNN, Multi-Dimension WNN (MD-WNN) and Filter-Multi-Dimension WNN (Filter MD-WNN) in RMSE for training data set with different hidden nodes

In Figure 5.13, it illustrates the comparison of prediction results of Support Vector Machine [81], Ripper[78], MD-WNN, Filter-MD-WNN, and AODE. The average precision, and the average recall in Figure 5.13. I use recall and precision to measure the effectiveness of our predictors.

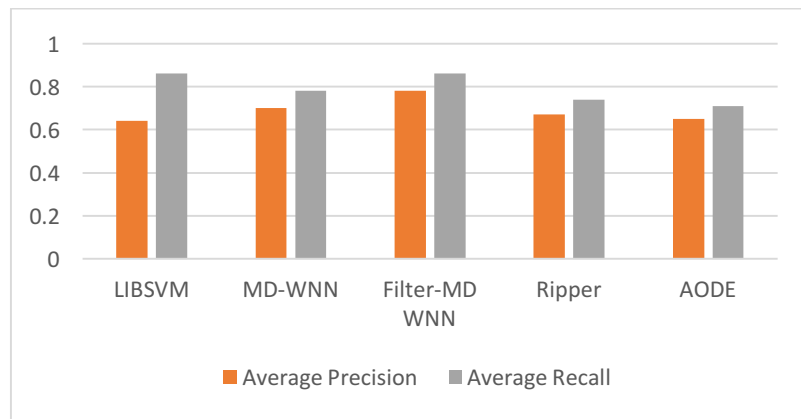


Figure 5.13 The average precision and recall of different prediction algorithms

The cost of time for computer the result, the different network settings and layers have outcomes. Comparing with SVM with the similar accuracy, the filter-MD WNN has a fewer time cost in the prediction process. The AODE uses the much more time to provide

the result. The time cost of Ripper is in between of LIBSVM and Filter-MD WNN. Although the MD-WNN has the least time cost, but the accuracy is worse than the Filter-MD WNN. The Figure 5.14 shows the comparison of the average time cost of different prediction algorithms.

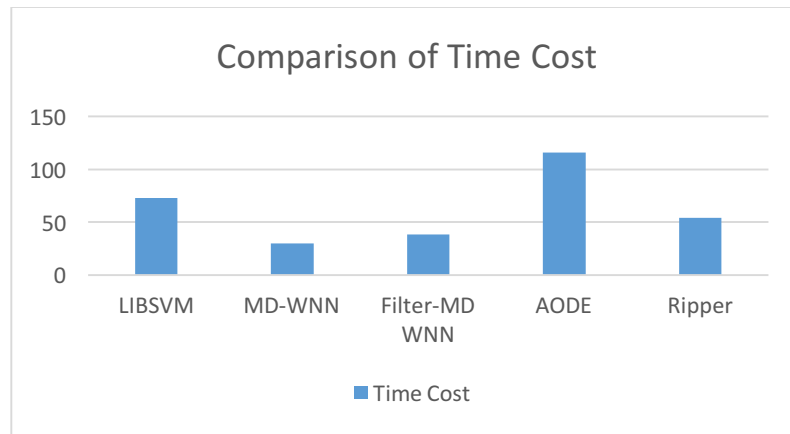


Figure 5.14 The average time cost of different prediction algorithms

5.5 Summary

In order to predict the cross virtual machine critical events, I proposed a multi-dimensional wavelet neural network based fast event pattern prediction framework. This framework combines features of hash table and reversed pattern tree structure to increase the learning and detection speed for real time critical event pattern. The reversed pattern tree provides relations between children nodes as well as children nodes and their parent nodes. This relation is used in prediction to increase accuracy. An attribute forest is also used for parallel detection and prediction processes. Primary attribute lists and secondary attribute lists supply additional information to increase the accuracy of prediction. The core prediction algorithm utilizes the multi-dimensional wavelet neural network to achieve a

balanced result between accuracy and speed. In order to improve the convergence, a filter layer is enforced to reduce unnecessary dimensions. Meanwhile, I find that the space complexity of real-time critical event pattern detection and prediction can also be reduced if a suitable set of wavelet parameters can be found.

CHAPTER 6: CONCLUSION

6.1 Conclusion

I have development customized agent architecture including policy analysis and service critical event prediction for cloud computing environment. This framework utilizes temporal logic to analyze configuration conflicts and predict potential service critical events. I developed knowledge-augmented temporal logic that incorporates semantic extension in a knowledge base to enhance logical expression and supplement reasoning capability. Experiments confirms the enhanced reasoning capability of this knowledge-augmented temporal logic and its excellence for multi-domain policy conflict analysis. Semantic extensions contain structural information of information domains, which include relationships and their related attributes. They provide the ability that I can incorporate different information domains without ambiguities. Dynamic relationships with constraints increase the accuracy of logic reasoning on changing information. This additional information can reduce the ambiguity of elements from different domains and relationships among multiple domains, which increases the accuracy of policy conflict analysis. Furthermore, semantic extension is flexible and extensible so that it can make collaboration and system integration easier. On the other hand, I developed a non-intrusion log-processing framework that learn and predict service critical events. This framework protects the cloud service through service critical event pattern learning and prediction. It uses a novel reverse pattern tree to store event sequences of collaborative services, and use filtered multi-dimension neural network online predict potential critical events. Therefore,

the static configuration policies and run-time service event monitoring are integrated into one customized agent architecture. The knowledge base ensures both the accuracy of policy analysis and critical event prediction.

6.2 Future Work

The reliability in cloud computing environment includes many aspects, the availability, data retention, real-time migration and other directions. These directions are all worth for working on. My future work will focus on using more run-time configuration policy analysis and conflict prediction using filtered neural networks in cloud computing applications.

REFERENCES

- [1] Clausing, Don, and Daniel D. Frey. "Improving system reliability by failure-mode avoidance including four concept design strategies." *Systems engineering*, vol. 8.3, 2005, pp. 245-261.
- [2] Rich Miller. "Software Bug, Cascading Failures Caused Amazon Outage." <http://www.datacenterknowledge.com/archives/2012/10/27/cascading-failures-caused-amazon-outage/>, retrieved by 2/28/2014.
- [3] Steve Cimino. "Cloud computing outages: What can we learn?" <http://searchcloudcomputing.techtarget.com/feature/Cloud-computing-outages-What-can-we-learn>, retrieved by 2/28/2014.
- [4] Chunye Gong; Jie Liu; Qiang Zhang; Haitao Chen; Zhenghu Gong, "The Characteristics of Cloud Computing," *Proceedings in 2010 39th International Conference on Parallel Processing Workshops (ICPPW)*, 2010, pp.275-279.
- [5] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C., "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol.1, no.1, 2004, pp.11-33.
- [6] Rich Miller. "Software Bug, Cascading Failures Caused Amazon Outage." <http://www.datacenterknowledge.com/archives/2012/10/27/cascading-failures-caused-amazon-outage/>, retrieved by 2/28/2014.

- [7] Zibin Zheng; Zhou, T.C.; Lyu, M.R.; King, I., "Component Ranking for Fault-Tolerant Cloud Applications," *IEEE Transactions on Services Computing*, vol.5(4), 2012, pp.540-550.
- [8] B. Randell and J. Xu, "The Evolution of the Recovery Block Concept," *Software Fault Tolerance*, M.R. Lyu, ed., pp. 1-21, Wiley, 1995.
- [9] A. Avizienis, "The Methodology of N-Version Programming," *Software Fault Tolerance*, M.R. Lyu, ed., pp. 23-46, Wiley, 1995.
- [10] Rosenblum, M.; Garfinkel, T., "Virtual machine monitors: current technology and future trends," *Computer*, vol.38(5), 2005, pp.39-47.
- [11] Matos, R.D.S.; Maciel, P.R.M.; Machida, F.; Dong Seong Kim; Trivedi, K.S., "Sensitivity Analysis of Server Virtualized System Availability," *IEEE Transactions on Reliability*, vol.61(4), 2012, pp.994-1006.
- [12] Jeffery, C.M.; Figueiredo, R.J.O., "A Flexible Approach to Improving System Reliability with Virtual Lockstep," *IEEE Transactions on Dependable and Secure Computing*, vol.9(1), 2012, pp.2-15.
- [13] Heeseung Jo; Hwanju Kim; Jae-Wan Jang; Joonwon Lee; Seungryoul Maeng, "Transparent Fault Tolerance of Device Drivers for Virtual Machines," *IEEE Transactions on Computers*, vol.59(11), 2010, pp.1466-1479.
- [14] Haibing Guan; YaoZu Dong; Kun Tian; Jian Li, "SR-IOV Based Network Interrupt-Free Virtualization with Event Based Polling," *IEEE Journal on Selected Areas in Communications*, vol.31(12), 2013, pp.2596-2609.
- [15] Jianhua Zhang; Wenbo Zhang; Heng Wu; Tao Huang, "VMFDF: A Virtualization-based Multi-Level Fault Detection Framework for High

Availability Computing,” 2012 IEEE Ninth International Conference on e-Business Engineering (ICEBE), 2012, pp.367-373.

- [16] Lei Cui; Bo Li; Jianxin Li; Hardy, J.; Lu Liu, “Software Aging in Virtualized Environments: Detection and Prediction,” 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012, pp.718-719.
- [17] Charalambides, M., et al., “Policy conflict analysis for diffserv quality of service management”. IEEE Transactions on Network and Service Management. vol 6(1), 2009, pp. 15-30.
- [18] Fagin, R., Halpern, J.Y., Moses, Y., Vardi M.Y., Reasoning About Knowledge. MIT Press, Cambridge, MA, 1995
- [19] Halpern, J.Y., Vardi, M.Y., “The complexity of reasoning about knowledge and time”. Journal of Computer and System Sciences. vol 38(1). 1989, pp. 195–237.
- [20] Halpern, J.Y., Van Der Meyden, R., Vardi, M., “Complete axiomatizations for reasoning about knowledge and time”, SIAM Journal on Computing. vol 33 (3). 2004, pp. 674–703.
- [21] Bacchus, F., Kabanza, F., “Planning for temporally extended goals”, Annals of Mathematics and Artificial Intelligence. 1998, 22. pp. 5–27.
- [22] Calvanese, D., Giacomo, G. De, Vardi, M.Y., “Reasoning about actions and planning in LTL action theories”. Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning. 2002, pp. 593–602.

- [23] Batt, G., Belta, C., Weiss, R., “Temporal Logic Analysis of Gene Networks Under Parameter Uncertainty”, IEEE Transactions on Automatic Control. vol 53, 2008, pp. 215–229.
- [24] Janicke, H., Cau, A., Siewe, F., Zedan, H., “Deriving Enforcement Mechanisms from Policies”, Proceedings of Eighth IEEE International Workshop on Policies for Distributed Systems and Networks. 2008, pp.161-172.
- [25] Jesper, G., Henriksen, P., Thiagarajan, S., “Dynamic Linear Time Temporal Logic”, Annals of Pure and Applied Logic. 96(1-3), 1999, pp. 187-207.
- [26] Beaudry, M.D., “Performance-Related Reliability Measures for Computing Systems”, IEEE Transactions on Computers, vol.C-27(6), 1978, pp.540,547.
- [27] Jhawar, R.; Piuri, V.; Santambrogio, M., “Fault Tolerance Management in Cloud Computing: A System-Level Perspective,” Systems Journal, IEEE, vol.7(2), 2013, pp.288-297.
- [28] T.F. Arnold, “The Concept of Coverage and Its Effect on the Reliability Model of Repairable Systems,” IEEE Transactions on Computers, vol. 22(6), 1973, pp. 251-254.
- [29] M.R. Lyu, Handbook of Software Reliability Engineering. McGraw-Hill, 1996.
- [30] Tsai, W.-T.; Xinyu Zhou; Yinong Chen; Xiaoying Bai, “On Testing and Evaluating Service-Oriented Software,” Computer, vol.41(8), 2008, pp.40-46.
- [31] K. Goseva-Popstojanova and K.S. Trivedi, “Architecture-Based Approach to Reliability Assessment of Software Systems,” Performance Evaluation, vol. 45(2), 2001, pp. 179-204.

- [32] S.S. Gokhale, "Architecture-Based Software Reliability Analysis: Overview and Limitations," IEEE Transaction on Dependable and Secure Computing, vol. 4(1), 2007, pp. 32-40.
- [33] A. Immonen and E. Niemela", "Survey of Reliability and Availability Prediction Methods from the Viewpoint of Software Architecture," J. Software Systems Modeling, vol. 7(1),2008, pp. 49-65.
- [34] R. Roshandel, N. Medvidovic, and L. Golubchik, "A Bayesian Model for Predicting Reliability of Software Systems at the Architectural Level," Proceedings on Third Int'l Conf. Software Architectures, Components, and Applications Quality of Software Architectures, 2007, pp.108-126.
- [35] Brosch, F.; Koziolk, H.; Buhnova, B.; Reussner, R., "Architecture-Based Reliability Prediction with the Palladio Component Model," IEEE Transactions on Software Engineering, vol.38(6), 2012, pp.1319-1339.
- [36] G.N. Rodrigues, D.S. Rosenblum, and S. Uchitel, "Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems," Proceedings on. Conference. Fundamental Approaches to Software Eng., 2005, pp. 111-126.
- [37] S.M. Yacoub, B. Cukic, and H.H. Ammar, "A Scenario-Based Reliability Analysis Approach for Component-Based Software," IEEE Transaction on Reliability, vol. 53(4), 2004, pp. 465-480.
- [38] Wei Chen; Toueg, S.; Aguilera, M.K., "On the quality of service of failure detectors," IEEE Transactions on Computers, vol.51(1), 2002, pp.13-32.

- [39] T. IDE and H. KASHIMA, "Eigenspace-based anomaly detection in computer systems," Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, 2004, pp. 440-449.
- [40] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," Proceedings on Fifth IEEE International Conference on Data Mining, 2005, pp.8.
- [41] S. Papadimitriou, J. Sun, and P. Yu, "Local correlation tracking in time series", Proceedings on Sixth International Conference on Data Mining, 2006. ICDM '06, 2006, pp. 456-465.
- [42] Ming-Da Ma; Wong, D.S.-H.; Shi-Shang Jang; Sheng-Tsaing Tseng, "Fault Detection Based on Statistical Multivariate Analysis and Microarray Visualization," IEEE Transactions on Industrial Informatics, vol.6(1), 2010, pp.18,24.
- [43] M.G. Gouda and T.M. McGuire, "Accelerated Heartbeat Protocols," Proceedings on 18th Int'l Conf. Distributed Computing Systems, 1998, pp.202-209.
- [44] R. van Renesse, Y. Minsky and M. Hayden, "A Gossip-Style Failure Detection Service," Proceedings on Middleware '98, 1998, pp. 55-70.
- [45] M. Raynal and F. Tronel, "Group Membership Failure Detection: A Simple Protocol and Its Probabilistic Analysis," Distributed Systems Eng. J., vol. 6(3), 1999, pp. 95-102.

- [46] Giordano, L., Martelli, A., Schwind, C., Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic*. vol.5(12). , 2007, pp. 214-234.
- [47] Zhang, J., Cheng, B.H.C., Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*. vol.79 (10), 2006, pp. 1361-1369.
- [48] Kolovski, V., Hendler, J., Parsia, B., Analyzing Web Access Control Policies. *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 677–686.
- [49] Goranko, V., Montanari, A., Sala, P., Sciavicco, G., A general tableau method for propositional interval temporal logics: Theory and implementation. *Journal of Applied Logic*, vol.4 (3). 2006, pp. 305-330.
- [50] Bowman, H., Thompson, S.J., A Decision Procedure and Complete Axiomatization of Finite Interval Temporal Logic with Projection. *Journal of Logic and Computation*. vol.13(2), 2003, pp. 195–239.
- [51] Moszkowski, B., A hierarchical completeness proof for propositional interval temporal logic with finite time. *Journal of Applied Non- Classical Logics*. 14(1–2), 2004, pp. 55–104.
- [52] Deng, L., Cai, Y., Wang, C., Jiang, Y., Fuzzy Temporal Logic on Fuzzy Temporal Constraint Networks. *Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 2009, pp. 272 – 276.
- [53] Eamani, A.K., Sistla, A.P., 2006. Language based policy analysis in a SPKI Trust Management System. *Journal of Computer Security*. vol.14(4). pp. 327-357.

- [54] Craven, R., Lobo, J., Ma, J., 2009. Expressive Policy Analysis with Enhanced System Dynamicity. Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. pp. 239-250.
- [55] Ahn, G., Xu, W., Zhang, X., 2008. Systematic Policy Analysis for High-assurance Services in SELinux. Proceedings IEEE Workshop on Policies for Distributed Systems and Networks. pp. 3-10.
- [56] McDaniel, P., Prakash, A., Methods and limitations of security policy reconciliation. ACM Transactions on Information and System Security. vol.9(3). 2006, pp. 259-291.
- [57] Agrawal, R.; Srikant, R., "Mining sequential patterns," Proceedings of the Eleventh International Conference on Data Engineering, 1995, pp.3-14.
- [58] Dunlop, N., Indulska, J., Raymond, K., Dynamic conflict detection in policy-based management systems. Proceedings of Sixth International Enterprise Distributed Object Computing Conference. 2002, pp. 15- 26.
- [59] Niksefat, S., Sabaei, M., Efficient Algorithms for Dynamic Detection and Resolution of IPSec/VPN Security Policy Conflicts. Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications (AINA). 2010, pp. 737-744.
- [60] Samak, T., Al-Shaer, E., Hong, L., QoS Policy Modeling and Conflict Analysis Proceedings of IEEE Workshop on Policies for Distributed Systems and Networks. 2008, pp. 19-26.
- [61] Wu,Z., Liu, Y., Knowledge-Based Policy Conflict Analysis in Mobile Social Networks. Wireless Personal Communications. 73(1), 2013, pp 5-22.

- [62] Roozmand, O., et al., 2011, Agent-based modeling of consumer decision making process based on power distance and personality. *Journal of Knowledge-Based Systems*. vol.24(7). pp. 1075-1095.
- [63] Kuhn, J.R.Jr., Courtney, J. F., Morris, B., Tatara, E.R., Agent-based analysis and simulation of the consumer airline market share for Frontier Airlines. *Journal of Knowledge-Based Systems*. vol.23(8). 2010, pp. 875-882.
- [64] Bellini, P., Mattolini, R., Nesi, P., Temporal logics for real-time system specification. *ACM Computing Surveys*. vol.32(1). 2000, pp. 12-42.
- [65] Liang, Yinglung, et al. "Failure prediction in ibm bluegene/l event logs." *Seventh IEEE International Conference on Data Mining, 2007. ICDM 2007*. IEEE, 2007, pp583-588.
- [66] F. Masseglia, F. Cathala, and P. Poncelet, "The PSP Approach for Mining Sequential Patterns." *Proceedings of European Symp. Principle of Data Mining and Knowledge Discovery (PKDD '98)*, 1998, pp. 176-184.
- [67] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining." *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD '00)*, 2000, pp. 355-359.
- [68] Derek Pao, Wei Lin, Bin Liu, "A memory-efficient pipelined implementation of the aho-corasick string-matching algorithm." *ACM Transactions on Architecture and Code Optimization (TACO)*, 2010, pp 1-22.

- [69] Wang, Jianyong, Jiawei Han, and Chun Li. "Frequent closed sequence mining without candidate maintenance," *IEEE Transactions on Knowledge and Data Engineering*, vol.19(8), 2007, pp.1042-1056.
- [70] Wu, Z., Liu, Y., 2012. "Knowledge-based policy conflict analysis for collaborative workspace," *Proceedings on 2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. 2012, pp. 727-734.
- [71] Friedman, N., Halpern, J.Y., Koller, D., "First-order conditional logic for default reasoning revisited," *ACM Transactions on Computational Logic (TOCL)*. vol.1(2), 2000, pp. 175 – 207.
- [72] Antony, G., "Temporal Logic," *The Stanford Encyclopedia of Philosophy* (Fall 2008 Edition), Edward N. Zalta (ed.).
<http://plato.stanford.edu/archives/fall2008/entries/logic-temporal/>
- [73] Artikis, Alexander, Marek Sergot, and Georgios Paliouras. "Run-time composite event recognition," *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. ACM, 2012, pp.69-80.
- [74] Bonchi, Francesco, et al. "ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints." *Proceedings of the third IEEE International Conference on Data Mining*, 2003, pp.11-18.
- [75] Campbell, R., Gupta, I., Heath, M., Ko, S., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H.Y., Lyons, M., Milojicic, D., O'Hallaron, D., and Soh, Y.C., "Open Cirrus™ Cloud Computing Testbed: Federated Data Centers for Open

Source Systems and Services Research", Proceedings of the USENIX Hotcloud'09, San Diego, 2009, pp.1-5.

- [76] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, F. Galán, "The reservoir model and architecture for open federated cloud computing." IBM Journal of Research and Development archive, vol. 53(4), 2009, pp. 535-542.
- [77] Yanlong Zhai, Jing Zhang, Kwei-Jay Lin, "SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints." Proceedings of the 2009 IEEE International Conference on Web Services, 2009, pp. 815-822.
- [78] M. Joshi, R. Agarwal, and V. Kumar. Mining needle in a haystack: Classifying rare classes via two-phase rule induction. In SIGMOD '01 Proceedings of the 2001 ACM SIGMOD international conference on Management of data, 2001, pp. 91–102.
- [79] Chen, Yuehui, Bo Yang, and Jiwen Dong. "Time-series prediction using a local linear wavelet neural network." Neurocomputing 69.4 2006, pp. 449-465.
- [80] Bebis, George, and Michael Georgiopoulos. "Feed-forward neural networks." Potentials, IEEE vol.13(4), 1994, pp.27-31.
- [81] M. Joshi, R. Agarwal, and V. Kumar. "Predicting rare classes: Can boosting make any weak learner strong?" In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002, pp. 297-306.

- [82] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proceedings of International Conference of Extending Database Technology (EDBT '96), 1996, pp. 3-17.