uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

HengHeng Xie

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.CS.

GRADE / DEGREE

School of Engineering and Computer Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A QoS-Aware System for Service Composition and Management on Overlay Network

TITRE DE LA THÈSE / TITLE OF THESIS

A. Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. El-Sadaik

G. Wainer

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A QoS-aware System for Service Composition and Management on Overlay Network

by

HengHeng Xie

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.Sc. degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-61275-0
Our file Notre référence
ISBN: 978-0-494-61275-0

Abstract

The research work of this thesis is motivated by a desire to build up efficient Quality of Service (QoS) insurance framework on overlay network. With the rapid development of Service Oriented Architecture, there is a growing interest in the design of efficient QoS-aware system. The difficulties of building QoS-aware system include service addressing, service composition, service management among others. Therefore, an integrated framework needs to be proposed to fulfill all these features. Meanwhile, load balancing is also a critical issue for solving scalability problem, when achieving high performance for large scale distributed systems due to the limited computing resource in general. In this thesis, we design and implement a JXTA Message Layer based infrastructure for improving the performance and functionalities of JXTA P2P framework, which can be used for designing better SOA based systems. Furthermore, a design of a QoS-aware hierarchical service composition and management framework is proposed and implemented on top of this infrastructure. A comparison experimental study of our design with commonly used flat-based service composition and management is conducted. The result of our experiments show that our design outperforms flat-based one in term of satisfaction of user requirements. A load balancing scheme is also proposed on this framework aiming to optimize the performance of our proposed framework on large scale distributed systems. We compare our load balancing scheme with non-migration and non-load-balancing approaches, and find that our solution based on a combination of task scheduling and service migration paradigm using Genetic Algorithm is much better than the other two in terms of QoS success rate.

Acknowledgments

I take great pleasure in thanking the many people who have helped me in my studies at the University of Ottawa. First, I would like to thank my supervisor, Professor Azzedine Boukerche, for his invaluable guidance and advice and a much needed financial support. I would like to thank Professor Gabriel Wainer and Professor Abdulmotaleb El Saddik for their careful reviewing on my thesis and serving on the exam committee, as well as Professor Shervin Shirmohammadi for serving as the chair of the exam committee. I also would like to thank Dr. Ming Zhang for giving me useful advice on my research area. Finally, I would like to extend my thanks to my parents and my colleagues, whose encouragement have always sustained me.

Contents

List of Tables

List of Figures

List of Publications

The following publications are relevant to the topic of this thesis and have been authored by Hengheng Xie.

Hengheng Xie, Azzedine Boukerche, Ming Zhang and Bernard P. Zeigler, 2008, *Design of A QoS-Aware Service Composition and Management System in Peer-to-Peer Network Aided by DEVS*, In Proceeding of 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'08), pages: 285-291

Azzedine Boukerche, Hengheng Xie and Ming Zhang. *A novel bio-inspired load balancing algorithm with QoS assurance for large-scale peer-to-peer systems.* Published in 42nd Annual Simulation Symposium (ANSS 2009), pages: 1-8.

Hengheng Xie, Azzedine Boukerche and Ming Zhang, *A Novel Message-Oriented and SOA based Real-Time Modeling and Simulation Framework for Peer-to-Peer System.* 2009 Winter Simulation Conference on December, Accepted.

Chapter 1

Introduction

Many researchers believe Service Oriented Architecture (SOA) can help processes response more quickly and cost-effectively to changing system structure. SOA provides an approach to encapsulate the software components as services, which are independent and interoperable. Most of these services perform specified basic functions. With the increasing requirements from service users, single service no longer satisfies their demand. In order to accomplish a complex function, service composition need to be introduced to SOA.

The technique of service composition is able to integrate services to collaboratively work on a larger task. However, service composition is unable to provide suitable or high quality services to users because it lacks the capability of identifying requests' requirements and matching them to services. Therefore, Quality of Service (QoS) need to be considered in a service composition system, which can provide a guaranteed level of performance to end users.

The provision of good QoS is one of the most challenging issues for efficiently attending requests' requirements, so many approaches have been proposed to solve such a problem. Although most of them attempt to provide high quality services path for each request, they fail to achieve the accomplishment of many requests.

Load balancing on a QoS-aware system is also a key factor affecting system per-

formance. Imbalancing workload may slow down the system running the services, and affect some specific QoS, like the response time. In order to guarantee a certain level of performance, one of the methods is to optimize the resource usage of the whole system. The addition of load balancing techniques to QoS-aware systems will be proved helpful.

In this thesis, we propose a tree-based QoS-aware service composition and management framework[45], which uses a JXTA-enabled P2P network infrastructure. We attempt to split the analysis workload through the tree structured system, while each branch can analysis part of the request. A proper service path will be offered for each request to achieve supplying suitable service path to all requests. A prototype of our design was implemented on a real JXTA P2P local network to test the performance of the system. Furthermore, we present our dynamic load balancing scheme on tree-based QoS-aware system in this thesis, a service migration solution for Genetic Algorithms. The service migration solution is designed for balancing the workload, on a lower level, among computer nodes.

This thesis is organized as follow: Chapter 2 briefly explains and surveys the different frameworks and techniques of the QoS-aware system. In this survey, we focus on the overlay network and the physical network of the QoS-aware framework. We also elaborate upon, to some extent, the existing load balancing techniques of QoS-aware systems. Finally, a few techniques used in our design, such as the JXTA P2P framework and Genetic Algorithm, are explained.

Chapter 3 refers to the design of the tree-based QoS-aware service composition and management framework. We present in detail the design of a tree-based framework, in this chapter, and the prototype implemented for testing the performance of tree-based frameworks. Several experiments are made to explain the advantages of our design over the commonly used flat-based framework.

In Chapter 4, we illustrate a load balancing scheme on the tree-based framework that we proposed in Chapter 3. This scheme is a service migration solution for Genetic Algorithm. We also refer to some experiments to demonstrate how our scheme benefits

the system.

Chapter 5 presents the design and implementation of JXTA Message Layer infrastructure. Moreover, here we suggest improvements to our design, and our implementation of the infrastructure.

In Chapter 6, we describe the implementation and experiments of our design in the previous chapters, to demonstrate the advantages of our scheme. The result shows that the Tree-based scheme can get a higher satisfaction of user requests in our experiments.

Chapter 7 concludes this thesis and suggest future areas of interest. The QoS-aware system has recently been a popular topic of study. There remains many avenues of possible development in this area. However, due to the constraints of this project, we can only briefly present areas concerning the QoS-aware system relevant to this thesis.

Chapter 2

Previous and Related Work

Many researchers have studied the QoS-aware service composition and management systems. Many approaches have been proposed such as [30, 4, 49], but most of them are based on web services[17]. The main disadvantage of web service are their poor performance and the issue of security. With the rapid development of peer-to-peer (P2P) techniques, the P2P system becomes another option for the QoS-aware system. In this chapter, several existing service composition methods are discussed for building QoS-aware systems on web service and P2P networks. Additionally, we briefly review a few load balancing techniques proposed by researchers on QoS-aware systems.

2.1 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) has become an emerging technique in recent years. Thus, many novel techniques have been proposed for improving the functional or non-functional features of SOA. However, due to increasing complexity in service-oriented systems, the current service-oriented system is difficult to design and organize. In terms of modeling a SOA based system, many researchers have proposed new techniques and methods [47]. These techniques typically can be classified as functional modeling and non-functional modeling. The purpose of functional modeling is to describe the func-

tions and processes of the system; this helps to obtain necessary information and to identify basic functions of the system. For instance, Sloane proposed a hybrid approach for modeling SOA systems using Colored Petri Nets (CPN) and MESA/Extend [38], which model the SOA system in two different layers: component level and black-box level. These two levels do not affect each other and can be validated separately. Kogekar presented a Middle-ware Building Block approach for SOA system modeling and performance analysis using Stochastic Reward Net (SRN)[29]. SRN is very close to designer's intuition, which is helpful for understanding the created model. Vale and Hammoudi described a Context-aware Service Oriented Architecture based on the Model Driven Development Approach[41], while, Krause demonstrated an approach to integrate existing system with intelligent models[31]. All three use semantics modeling ontology to describe web services.

On the other hand, non-functional modeling attempts to illustrate the specified criteria used to judge the operation of a system without focusing on its behaviors. For instance, Quality of Service (QoS) is one kind of non-functional requirements, which provides guaranteed performance to users. With regard to research for modeling the non-functional aspects of SOA systems, Wada proposed a non-functional modeling approach with a UML profile [42]. This method can specify and maintain the SOA's non-functional feature through a manner of implementation independence. Another new Model Driven Development (MDD) framework was later presented to model the non-functional constrains of the SOA system[43]. Combined with the consideration of QoS, Zhiang and Junzhou proposed a QoS-Resource Graph Model (QRGM) for modeling end-to end QoS [44].

It's worth noting that the above surveyed approaches focused on modeling SOA system design. However, they cannot be used to directly model an implemented real SOA system. In this thesis, we propose a novel SOA based development tool built upon a JXTA overlay network, which can effectively model real SOA implementations. We build a Message Layer on top of our P2P service framework to support high level SOA

modeling. The virtual network built on JXTA in our framework is shown on Figure 2.1, in which a set of JXTA peers can be organized as a group where they communicate with each other within this same group. Peers in different groups can exchange messages only through some special peers.

2.2 Quality of Service (QoS)

Quality of Service (QoS) first appeared in the field of telecommunication networks, to provide guaranteed performance levels to user, application or data flow. In the telecommunication field, networks or protocol set different priority level for different users, applications and data flows, therefore identifying different requirements. With the dramatic increase in the amount of users demanding popular services and complexity of service work-flow, QoS become one of the most important methods to satisfy most users, especially while bandwidth and computing capability was limited. Some researchers may consider the research to increase bandwidth and computing capability more valuable than that research performed on QoS. However, there is always a limitation on computing resources. QoS is able to maximize resource usage, in order to benefit as many users as possible.

2.3 Review of QoS-aware Systems

The main purpose of QoS-aware system is that users can conduct complexly composed services transparently. QoS-aware systems create an abstract view of work-flow for users. In order to fully support this feature, the system should be (semi) automatic and self-organized. In this section, we will introduce some existing QoS-aware framework, which can provide some background information for our research.

2.3.1 Service Composition Algorithm

Web Service Business Process Execution Language (WS-BPEL)[36, 10, 11], BPEL in short, is one of the most popular open standard executable languages for describing the interactions of web services. WS-BPEL is an orchestration language based on XML language. It uses XML to describe the interaction of services and to specify the messages between business processes and other systems. The goal of BPEL is to define messages transferred to external processes or systems by WSDL and to provide orchestration concepts of the business process for web service composition.

Service composition results in BPEL are defined in two ways: executable business process and abstract business process. Services that participate in this processes are called partners. Messages exchanged among services are call activities. BPEL serializes processes into XML, and integrates different services into a large and complicated program. BPELJ, another version of BPEL has been released for the integration of BPEL and Java. This version allows users adding executable Java codes in BPEL to finish specified operations. However, security issues are always a significant subject for web service. Though BPEL specifies web service composition in XML language, it does not change or improve security techniques for utilizing HTTP.

Other approaches have been proposed to support QoS techniques on web services. [46, 16] introduce and discuss service composition approaches, which are based on work-flow analysis and management techniques. They are both motivated by the work-flow language BPEL that we introduce in the last paragraph. [46, 3, 26] assert that BPEL and almost all other existing techniques are centralized on the work-flow policies, which may limit the system. This paper proposes an approach to add Peer-to-Peer (P2P) interaction during the service process. When services need to communicate in an asynchronous way, they communicate in P2P channel. This approach splits the work-flow into different parts and lets them work in an asynchronous channel to improve the performance of the system. [16] focuses on work-flow discovery and the ranking problem. It first discusses criteria and requirements that a work-flow discovery tool should consider. The author

then describes the implementation of their proposed design on work-flow discovery as a plug-in to the Taverna Workbench based on the criteria and requirements presented in the paper. Different algorithms than BPEL or other orchestration frameworks also contribute to the service composition area. [8] presents a study on solving service composition problems by using the Genetic Algorithm. This approach quickly determines a set of concrete service related to the abstract service of work-flow. It first encodes the problem into a genome for the Genetic Algorithm to analyze, while the genome is classified by the type of abstract services. Based on the Genetic Algorithm, this algorithm is able to create a set of generation, which is distinct from each other. In order to select a specified genome from the set of generation, this paper proposes a fitness function to eliminate the unsuitable genome. This fitness function also adds a dynamic penalty factor to increase the surviving probability of the solution.

Some researchers consider the service composition on web service as a graphic problem. Researchers study the graph problem in order to solve the problem of services selection. [33, 40, 39] proposed a web service composition framework, which is extended from Triana, an open source problem solving environment. In their design, users are able to share their composing service as a BPEL4WS graphs. Users publish their composing services to a UDDI server, so other users can query these services by keywords. The major problem for this framework is that it's not automatic. Users need to create their own service selection. [44] presented a graph modeling technique for web service composition. In this paper, the qualities of services will be represented as vectors for comparison and reference. Nonfunctional properties are also defined in this framework, like security, dependability and performance. However, it only models a single service in this paper. [2] proposed a service composition framework on MAIS (Multichannel Adaptive Information System), which supports access to information system through different devices. In this paper, it assumes all tasks have only one input and one output. Based on this assumption, this framework is able to handle the loop process in the abstract execution path. Authors describe the web services selection problem as a mixed integer linear

programming (MILP) problem to decide how to concretize the abstract services path. If there is no possible solution for the abstract execution path, a negotiation process will be conducted between broker and service provider.

Though web service is a formal and popular approach for providing services to users, many existing problem limit the usage of web service, such as security and addressing problems discussed in the previous section. Most of the service composition framework proposed for web service is through the orchestration concept, because the publishing and discovery process on web services rely upon UDDI. Some of them are also unable to self-organizing and monitoring, and some may have difficulty to self-organizing and monitoring, because normal web services are stateless and limited in terms of capability. However, some algorithms for service composition on web service can be improved or revised to use for general purposes.

2.3.2 Distributed and Hierarchical

Many more techniques on service composition have been proposed for the QoS-aware system on Peer-to-Peer (P2P) networks or for general purpose, than those for web service that we discuss above. Some of these proposed techniques are based on fully distributed structures, while others are hierarchical. For example, SpiderNet[18] is one of the most popular QoS-aware frameworks for the distributed structure system. Both the distributed structure and the hierarchical structure have advantages and disadvantages. The distributed system is flexible and fault tolerant, but it's difficult to monitor and manage. In contrast, hierarchical system have a global view of systems and manage systems with ease. However, hierarchical systems are difficult to maintain and implement fault tolerance. In this section, we will discuss existing techniques for QoS-aware systems in both distributed systems and hierarchical systems on Peer-to-Peer networks.

SpiderNet [18, 20, 19] is a QoS-aware service composition, for large scale system, that operates on a peer-to-peer overlay network. There are three main feature of SpiderNet.

It is able to support dynamic services selection, while most existing techniques only support fixed work-flow. Secondly, it also supports a peer's ability to join or leave the system dynamically without affecting the system. Moreover in order to be practical in a large scale system, all services do not need to maintain or retrieve the global view of the whole system in Spidernet.

The service path is represented as a chain of service components. This research also proposes a model for service components to ensure the QoS requirement. There is a separate set of quality properties for both the input and output of each service. A service accepts only the output from its previous services, when its quality properties meet the requirements of the input quality properties. Thus it ensure that no bottleneck exists in the service path and that the quality of the whole service path will satisfy the requirements of users.

This framework includes the following major functions: service path selection, service path instantiation and benefit driven peer clustering. The service path selection is mainly based upon user requirements, and selects a collection of available services from redundancy services in the system. To achieve a path selection, it needs to first acquire the meta-data for all services referred to as user requirements. After retrieving this information, it composes a service chain based upon the input and output quality properties of different services. If there are existing redundancy service paths, it calculates the resource usage of all candidate service paths, and selects the shortest one by Dijkstra's algorithm. When the service path has been decided, it will be instantiated to the proper peers in the system and will begin processing the user request. In order to improve the performance of the system, a benefit driven peer clustering mechanism has been proposed. Each service maintains a list of useful neighbors selected during the service path selection procedure. This service also perform a periodic probing of useful neighbors to measure their performance. This information is used in the later service path selection.

In SpiderNet, each service probes the functionalities and QoS properties of their

neighbor. They can make decision to select the next-hop service component. This solution prevents the overhead caused by centralized management. SpiderNet also supports failure recovery through the backup knowledge of service graphs. However, there are still some drawbacks of SpiderNet. Each service is only visible to the neighbor service; therefore it is difficult to manage and integrate global resource in SpiderNet. Though SpiderNet is a distributed framework, it still needs to collect basic information of all services, when initializing the process of a receiving request.

Binary Search Tree based Solution [37] organizes the system as a binary search tree for increasing the speed of service searching and selection. Most hierarchical systems maintain the service information on server side with a centralized information agent. In contrast, this framework maintains the binary search tree on the client side. Consumers will save the service requested and use as a binary search tree for QoS information. In a binary search tree, the best service is the the right most service node of the tree. To use this node, a service registry agent is set up to collect the information of consumers. The service registry also manages the information of the service. Both consumers and service providers need to register to the service registry. The centralized service registry is able to collect global information about consumer information and the service requested by consumers.

If a consumer requests a service from the service registry, the service registry searches the information from other consumers, who previously requested the same service. If there was someone using this kind of service before, the service registry copies this service tree information to the new requester. Otherwise, the service registry searches the registered services and send a set of the existing proper services to the requester. Based on these services from the service registry, consumers will build their own service tree and select one of the services. After using the service, an update of the QoS properties is sent from the service requester to the service registry. Service registry identifies the consumer and the service and records the QoS update of the service.

The advantages of a binary search tree based solution are the rapid service search and selection, separating the analysis workload on the client side and the global management of consumer and services. However, the centralized service registry also has a limited performance, because it causes overhead when the number of services increase. Another drawback of this solution is that it only supports single service selection.

2.3.3 Summary

As we discussed in the previous paragraph of this chapter, the existing shortcomings of web service are poor performance and lack of security. This is one of the major reasons we prefer to perform our research on a general P2P system, without using web service techniques. We have discussed several existing service composition algorithms in section 2.3.1. Many techniques have been proposed on work-flow analysis, graphical solution or other algorithms. Researchers try to find an efficient way to determine a composed service, which can fully or even partially satisfy the requests of consumers. Though most of these solutions are based on web service techniques, some of them can be revised or improved to support general services on a P2P system.

In section 2.3.2, we compare the advantages and disadvantages of a distributed system and a hierarchical system, by presenting two examples, SpiderNet and the binary search tree based solution. Both the distributed system and the hierarchical system have benefits and shortcomings; each system needs to integrate with other techniques to improve their performance and functionalities. In this thesis, we design our framework on a hierarchical solution, because we want to develop a system that is adept at monitoring and managing services in order to provide properly composed service to requesters. In our design, we try to supply proper services to requesters, to satisfy as many consumers as possible.

2.4 Load Balancing Techniques on QoS-aware Systems

With the rapid development of Quality of Service techniques, there is an increasing demand for service that can satisfy specified user requirements. However, most QoS-aware systems cannot solve problems caused by the increasing amount of requests. For instance, a system may crash or slow down, when many requests rush in and cause as imbalanced workload distribution. Some QoS properties, sensitive to workload, may be affected, such as task execution time and system response time, which lead to QoS failure. To solve this problem, load balancing techniques should be introduced to the QoS-aware system. In this section, we will present some previous studies on the load balancing techniques of QoS-aware systems.

2.4.1 Routing Algorithm and Traffic Engineering

Most load balancing techniques for QoS-aware systems appear to focus mainly on routing algorithm and traffic engineering. This research treats and attempts to optimize the usage of limited bandwidth and computing resources. In order to support high quality services for as many requests as possible, efficient traffic management and routing algorithm are essential. [28, 27] propose an efficient bandwidth management algorithm for online multimedia service providers. This framework balances the traffic load among different multimedia services, and guarantee efficient bandwidth usage. The techniques supported by this framework are bandwidth reservation, call admission, bandwidth migration and call-preemption strategies on QoS-sensitive multimedia networks. Bandwidth reservation is one of the most efficient techniques for hand-off, when users change their position in the cellular mobile network. To provide dynamic bandwidth reservation, the reserved bandwidth shares different priorities with some conditions. A time-driven component is also created to consider both the traffic history and recent traffic utilization, when adjusting the amount of reserved bandwidth. In order to balance the amount of available

bandwidth among the cell, the framework can achieve bandwidth migration between cells. Specific cell can lend bandwidth to a heavily loaded cell, in order to balance the traffic load among cells.

Ronald's studies [21] on the Ethernet routing proposed an Ethernet-specified load balanced (ELB) routing mechanism, to provide robust dynamic traffic demand . This paper also compares ELB techniques with the multi-protocol label switching (MPLS) load balancing (MLB) mechanism, one of the most popular technique for delivering Ethernet services. This mechanism creates a spanning tree for each node, providing the shortest paths from their root bridge to all other node in the network. By comparing the ELB technique to the MLB technique, it demonstrates how ELB is able to reduce the propagation delay and bandwidth utilization. This illustrates that ELB can support more traffic transportation than MLB. In addition, MLB only supports congestion paths between specific nodes, whereas ELB supports congestion paths between every node. For traffic reduction when broadcasting and multi-casting, ELB copies only the frame when a divergence occurs. However, MLB copies frames for different destination nodes.

In order to optimize the performance of traffic engineering, [35, 34, 32] proposed a new traffic control law that simply requires the source inferrable information for congestion detection. This algorithm does not require real-time feedback from the network outside the source. This paper assumes that the only considered resource in the network is the linking bandwidth, which corresponds to the fluid flow model. The purpose of this paper is to provide a decentralized solution to maximize the utility function in such a network model. This algorithm can solve the routing problems of both the point-to-point multi-path connection and point-to-multipoint multi-path forwarding. Because this algorithm is designed in a decentralized framework and all the nodes run with a set of control laws independent of each other, this algorithm is good for scalability and fault tolerance.

Iftekhar Ahmad and Joarder Kamruzzaman [1] present a mathematical formulation for preemption aware QoS systems based upon the current instantaneous request (IR) calls and future book-ahead (BA) load information. Both IR and BA are resource reser-

vation techniques for QoS requirements of high speed multimedia and distributed application. The difference between IR and BA is that IR request immediate resource for reservation and usage, while BA reserves resource in advance. Because BA reserves resource in advance, an IR may preempt resources reserved for a future BA call in a heavily loaded link. This algorithm tries to solve the preemption problems and to reduce the probability of preemption, while using BA and IR techniques. This algorithm creates a BA table for each individual link to record the BA reservation information. This BA table can be used as a reference for IR reservations. A preemption policy is also proposed for solving the conflict of reservation.

Both routing algorithm and traffic engineering are the basic functions of traffic load balancing on QoS-aware systems. The purpose of these techniques is to balance the traffic load over the network so that the performance of the whole system may be increased. However, most of these traffic techniques are only suitable for distributed systems, which are very difficult to revise into hierarchical frameworks. Traffic engineering is a load balance technique on the link level, and many excellent solutions have been proposed. We would like to study the problems concerning the application level and perform research on the relationship among services and the relationship between the request and service.

2.4.2 Other Load Balancing Schema for QoS-aware System

Besides the traffic engineering and routing algorithm many load balancing schema have been proposed for the QoS-aware System, in order to balance the workload over the system. In this area, many researchers study the behaviors and relationship of services to optimize their performance. Most of the QoS-aware research does not consider load balancing, when designing the behaviors of the system. They only consider the optimized solution for one request. However, it's impractical for a QoS-aware system to not consider load balancing. We will present information on the load balancing scheme in this section to illustrate the background information of our research.

For providing realistic 3G services traffic classes and user prioritization, [25, 12, 13]

provide research on the integrating geography load balancing technique with the QoS-aware system. With the geography load balancing technique, this framework supports dynamic base station (BA) coverage in its ability to provide service to high priority users on a wireless network, when the corresponding BA is busy or unable to provide such services. Thus in this framework, the blocking rate of high priority users is decreased by transferring the workload from one BA to an adjacent BA. Without dropping any connections, this framework supports more users by integrating with the geography load balancing technique, which make this framework more adaptive normal from QoS-aware system.

In order to completely design a scalable content-aware load balancing solution for the cluster-based web service system, [15] presents a load balancing algorithm, named Adaptive-Predictive with Resource Allocation (APRA), for lowering overhead caused by the algorithm. To design a dynamic solution for different situations, the size of a time slot and utilization level reservation may be modified based on the current decision. This algorithm is also able to estimate the future workload that a service will receive in the next time slot; it can therefore make a decision concerning a request assignment based on this prediction. This algorithm classifies the request into two major types, static and dynamic. Static requests only require access to web services, whereas dynamic requests require access to the database and back-end services. These two types of services need different workloads for processing. This algorithm considers both in different utilization levels. For each service, this algorithm records the previous throughput prediction and the throughput measurement of a current and previous slot. Based on this information, the algorithm predicts the throughput of services in the following slot.

Feng Zeng and Zhigang Chen [48] propose a load balancing algorithm for gateway placements in wireless mesh network with QoS constrains . When the traffic in a wireless mesh network become very heavy, the gateways may become the bottleneck of the whole system. It may also affect the QoS performance in the system. However, the placement of the gateways may improve the situation. This paper proposes a greedy algorithm for

load balancing clustering (GA-LBC). First, there is an input of initial gateway nodes in the system; the system is, in turn, divided into disjointed cluster nodes. This algorithm then will check the coverage of nodes. If some of the nodes are not covered by any gateway, an uncovered node will be selected randomly as a gateway to cover more nodes. These procedures will stop, when all the nodes have been covered. However, this greedy algorithm can only solve the coverage problem. To find an optimal solution, authors also propose another hybrid algorithm for the load balancing placement of gateways (HA-LBPG); this integrates both with the genetic algorithm and the greedy algorithm. The input of the genetic algorithm is the computing result of the greedy algorithm, which is a valid clustering. In the selection step, this algorithm uses a roulette wheel selection algorithm to select two individuals for the next generation. These two individuals are the parents of next generation. The parents will generate new individuals in the crossover step, in order to preserve good clusters and to eliminate the bad clusters. In the mutation step, some nodes in a specified individual will change their status, possibly from that of a normal node to a gateway in order to create difference within the population. The genetic algorithm enables the nearest optimal solution to be found efficiently.

2.4.3 Summary

Though many load balancing schema on the QoS-aware network have been presented in section 2.4.1, most of them focus on low level connection, like the routing algorithm, traffic engineering and bandwidth control. However, they do not consider the impact to QoS satisfaction caused by workload change on services. We try to examine the behavior of services in this thesis, and explore the relationship among services and the relationship between requests and services. Some other techniques have been proposed in section 2.4.2 based on the study of the service. Though these researches optimized the assignment of requests, the research does not consider the computing capability of nodes that service is running on. In this thesis, we proposed a load balancing scheme to balance the requests' workload over the services and also to balance the services on computing resources.

2.5 JXTA

In our research, we used several techniques to support our work. We used JXTA to build the peer-to-peer overlay network infrastructure, which is a mature and powerful P2P framework. JXTA has several fantastic features, such as its publishing and discovery protocol. We made good use of these features to enhance the performance and capabilities of our system.

JXTA[23, 24] is a mature Peer-to-Peer framework, which can provide efficient communication among different devices. JXTA provides a set of protocols for different devices to communicate as mature peers over the network, which is an open source project. JXTA is a programming language independent tool. It has been implemented on Java platform, C, C++ and .NET. We will use Java as the standard programming language in our research. Consequently, we choose the JXTA Java that is implemented on Java platform.

The major advantages of the JXTA technique are interoperability, platform independence and ubiquity. JXTA allows devices to discover and communicate with each other without noticing the network address and physical protocol. It's an overlay network built on top of the physical network. With JXTA, peers are able to locate each other and exchange messages simply with an unique ID assigned by JXTA protocol. Even if a peer moves over the network, changes the network and the transport address, and is disconnected from the network for a period of time, other peers can locate this peer with JXTA ID. Thus JXTA is not only independent on programming languages, but also independent on transport protocol. All devices are able to communicate with each other with JXTA. Figure 2.1 is an example of the JXTA framework.

There are several components of JXTA that enable users to develop networked, interoperable applications. The components are as follows: peers, advertisements, peer groups, sockets and pipes. Moreover, these components are identified by JXTA IDs, which are unique in the network. Peers can be any devices in the network or even a vir-

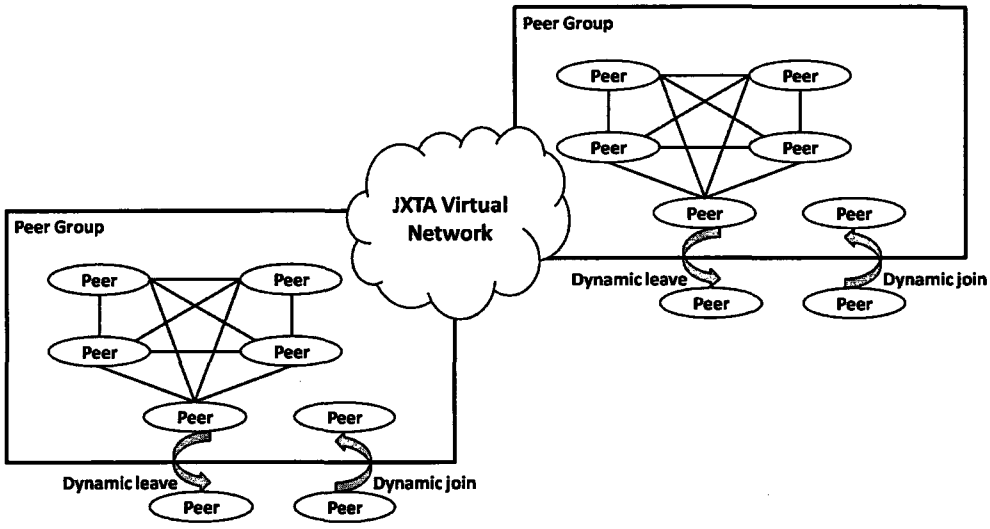


Figure 2.1: An Example of JXTA Architecture [5]

tual process. Therefore several peers can exist in one physical device. In contrast, several devices also can act as one peer. Common peers can be classified as Minimal-Edge Peers and Full-Edge Peers in terms of their functionality. Several peers act as super-peers to maintain the global connection and service discovery in the network. There are three types of super-peers, Relay Peers, Rendezvous Peers and Proxy Peers. Relay Peers store and forward messages between peers that do not have direct connection like peers blocked by a firewall. Rendezvous Peers maintain global advertisement indexes to help other peers index advertisements on a different subnet. Proxy Peers are used to encapsulate messages from Minimal-Edge Peers, enabling them to access all functions of the network.

Peers can be organized into peer groups, which restrict the connections of peers. All the peers will join the Network Peer Group by default. Each peer is able to join several peer groups for different purposes. Peers are used to provide services or resources in two ways: peer services and group services. Peer service is supplied by one peer. Group service is provided by a group of services, which has replicated services to supply the same type of service. Group service may also be provided through cooperating service

that supply one service together.

In order to connect these services together, all kinds of networking protocol, such as TCP/IP, HTTP, Bluetooth and GSM, are used. After peers are connected with each other, they will locate other peers through JXTA advertisement, which is an XML document, a programming language independent document. First, services and resources need to publish the advertisement for other services to discover them. The advertisement contains information about contact information of nodes. In order to reduce the time for discovering the advertisement, there is a cache of discovered advertisements. When peers want to discover a service, they will search the cache first. If the requested service advertisement is in the local cache, peers will reuse this advertisement. If not, peers will send a remote discovering request to search for new services. After retrieving the advertisement, peers build pipes or sockets based on the information in the advertisement, thus establishing the physical connection. Pipes are asynchronous and uni-directional communication channels, while sockets provide bi-directional connection; each pipe has to connect to the endpoints of peers.

2.6 Genetic Algorithm

We used Genetic Algorithm to make decisions for the load balancing scheme. The Genetic Algorithm is well known to be efficient in finding the nearest optimal solution for multi-goal problems in [9]. This is the reason for which we used the Genetic Algorithm to optimize our workload balancing of the system.

The Genetic Algorithm [9, 22] is a numerical searching technique used to find exact or approximate solutions for optimization and searching problems, and is inspired by natural selection and evolutionary biology. The Genetic Algorithm was invented by John Holland in the 1960s to provide a novel and thorough means of determining future behavior, with an interest in optimizing the expected reward. Coley [9] illustrates that a typical genetic algorithm should include four components: 1) a number of possible solutions

for the problem; 2) a mean to measure the solutions for the problem; 3) an approach to exchange the fragments of a good solution to create a new and better solution; 4) a mutation operator for adding diversity to the solution. An algorithm that contains these four basic components can be call genetic algorithm.

Coley [9] also demonstrates how to use Genetic Algorithm to search problems. Unlike the original search algorithm that starts from a single point in a searching space, Genetic Algorithm prefers to create a set of possible solutions guessed for the problem; this is usually called the population of solution. Typically, each solution is encoded as a string or binary. The typical genetic algorithm has three main operators for processing the population of the possible solution: selection, crossover and mutation.

Selection The selection operator functions to eliminate a number of the population from the possible solution. It resembles the natural selection process, which is well known in the field of biology. Individuals with poor performance have a greater danger of elimination by the forces of environment. The better ones have a greater chance if survival. This process tries to preserve as many better individuals as possible and to promote them to the next generation. First, a fitness function evaluates the survival possibility of each individual. A survival possibility may be calculated from different aspects with weight. The selection process chooses a set of individuals based on their possibility for survival and promotes them to the next generation. There are two types of commonly used selection algorithms, tournament selection and roulette-wheel selection.

Crossover Crossover is a genetic operator that produces various individuals from one generation to the next generation. It's similar to the natural reproduction process of species. Parents generate a set of children, which contain a gene consisting of a series of chromosomes partially from each parent. All or part of the remaining individuals from the previous selection process join the crossover process for generating the next generation. Participating individuals exchange parts of their chromosomes to generate new individuals based on specific algorithms. There are several popular crossover algorithms,

some of which are one-point crossover, two-point crossover and Cut and splice.

Mutation To avoid permanent loss of diversity, the mutation operator is processed after selection and crossover. The mutation operator is analogous to biological mutation, where the sequence of chromosomes in an organism are changed. The purpose of mutation is to create a variation within the gene to consider all possible solutions. After some steps in selection process, some specified chromosomes may be eliminated from the generation, which can be useful or useless. Mutation can bring them back to generation for reconsideration. Another reason for operating mutation is that after several generations the population of chromosomes will become too similar; this slows down or even stops evolution.

After these three operations, the counter for generation will increase by 1. The evolution process will start again with this generation as an initialized population. To stop the evolution process, a maximum number of generations can be set, or other criteria must be met: for example a required individual is found.

Chapter 3

Tree-Based QoS-aware Algorithm

QoS-aware management and composition, and its applicability to SOA architecture, has become a very popular topic. We have presented several existing service composition techniques in section 2.3.1. As discussed in section 2.3.1, web services are ideal for building large scale service composition frameworks, due to their lack of efficiency and security issues. Thus, we would like to design a service composition system on a Peer-to-Peer overlay network. With the rapid development of the P2P network, many P2P frameworks and tools have been proposed and developed. We chose JXTA, which is designed and developed by Sun. JXTA is a mature P2P framework supporting efficient peer communication. JXTA is programming language independent tool, which makes it interoperable on different platform. The major feature of JXTA is its discovery protocol. All components in JXTA can be discovered by advertisements, containing the component's connection information. Other peers can connect to this peer, based on the advertisement without noticing the network address. Our system will become more flexible with this feature. This is the primary reason we chose JXTA as our network infrastructure.

Two major structures of the system exist: the distributed and hierarchical system. Both kinds of systems have their advantages and disadvantages. Distributed systems are flexible and easily maintained, because peers in distributed system are self-organized or

semi-self-organized. However, for peers it is difficult to have a global view of the system, and the cost for a global view may be very high. Hierarchical systems do not have such problems. There may be a manager for monitoring the system and organizing the other peers. With hierarchical systems problems arise if the manager becomes a bottleneck , or if the failure of peers causes the system to crash. We tried to design a system, which can determine the proper solution for the request. The solution provided may not be the best solution, but it can satisfy the request. A better solution can be preserved for other requests with higher requirements. Thus, our design is able to monitor the resources of the system, and to make decisions for incoming requests. Therefore, we chose a hierarchical structure for our system, which is suitable for our system requirements.

In this chapter, we present our design of a tree-based hierarchical QoS-aware service composition and management system. Our design philosophy is to best manage and compose service components in real time according to the functional and QoS parameters provided by the applications. Our service composition and management scheme is based on three items: efficiently searching for the proper service path through tree-based service composition and management, balancing the workload of service management and adapting to the real-time service components' changes that include the QoS-serving capabilities

3.1 Architecture

There are many kinds of topology, such as bus topology, ring topology, star topology and tree topology. We chose tree topology to design our research; this topology is also known as hierarchical structure. In tree structure, each node only notices its parents and children. We will use this feature to separate the workload among the nodes. However, normally the nodes on higher levels of the tree need to process more data than these on the lower levels. In our research, the main workload is caused by the analysis of request. Our design is able to balance the analysis workload from one level to another. As shown

in Figure 3.2, we designed a tree-based service composition and management system, and then implemented it in a JXTA-enabled P2P network environment. All service components were grouped dynamically according to their QoS-serving capabilities and functional classifications. In order to consist with the programming language independent and platform independent features of JXTA, all the messages exchanged among the components of our system were in XML form.

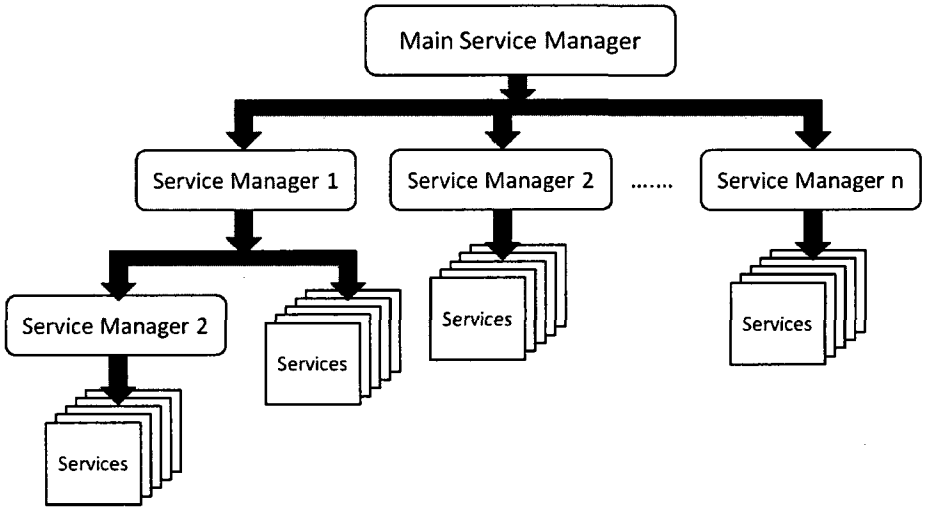


Figure 3.1: System structure of Tree-based QoS-aware system [45]

There are two major components in our system, service managers and services. The basic functions of service managers are to analyze and to assign requests. A service manager can manage both the service and other service managers. A request is treated as a task in our system. Upon the reception of a request, the system will encapsulate it as a task for further analysis and assignment. Each service manager analyzes part of the task and assign part of the task to the service or other service managers based on user QoS requirements. The major function of services is to process tasks assigned by service managers.

In our tree-based framework, the main service manager assigns tasks to its children service managers according to a task's functional and QoS requirement. The composition and the layout of these service managers depend on the internal structure of a specified

task and the dependency of the task's sub-components. All service managers are implemented as JXTA services, they discover and communicate with each other through a built-in JXTA discovery protocol. Due to this hierarchical form of service management, each manager only knows the type of services supplied by its children managers in addition to the QoS-serving capabilities of its local services. In other words, the details of services are unknown if these services are not immediately attached to a service manager. In fact, each service manager maintains a list of all its children managers, as well as the type of functions they provide. When a task enters the main service manager, the main service manager directs the task's requirement to all children managers with the necessary services to complete the task. Each child manager then evaluates the requirements to determine if these can be met through the use of a local service path. If the path is locatable, the child manager responds to the parent manager with an "OK" message; the parent manager then assigns the task to the child manager who responds most promptly with an "OK" message. Conversely, if children service managers cannot find any local service paths allowing them to meet task requirements, they forward the requirement to their children managers until a service path can be established.

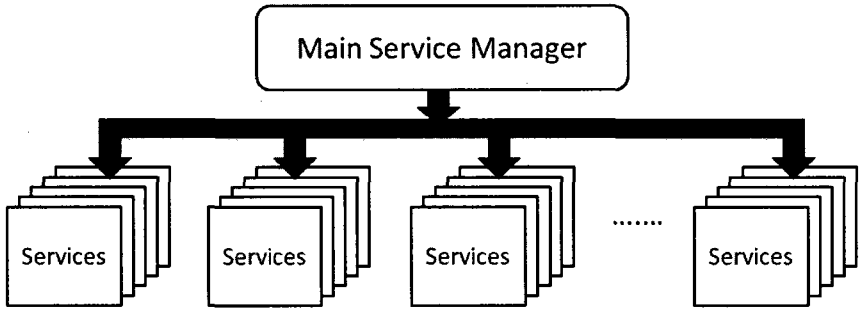


Figure 3.2: System structure of Flat-based QoS-aware system [45]

In contrast, we also implemented a flat-based service composition framework, as shown in Figure 3.2. We compared this flat-based service composition framework to our Tree-based framework in terms of user satisfaction, represented as the QoS success rate. In the flat-based framework, there is only one service manager, the Main Service Manager;

it is connected directly to all available service components. The Main Service Manager must therefore analyze the entire task tree and locate suitable services and service paths from the whole service list. The Main Service Manager splits the task tree into several execution paths, and sends the paths to the chosen services. These services can thus work with the paths without interacting with the Main Service Manager. When one service finishes its job, it sends the result to the next chosen service, which is determined by the information stored in the service path. The comparison of these two frameworks demonstrates the advantages of our system structure. Though the basic function of these two frameworks is similar, the system structures are different.

3.1.1 Task Scheduling

When facing a huge amount of incoming requests, it's important to distribute requests to all possible services equally, especially when QoS properties can be easily affected by work load. Some examples of QoS being sensitive to work load are online TV, online games, emergency systems, economic transactions and so forth. Normally, each service has a task queue and executes tasks one by one. When one task occupies the services, other tasks in the queue are put on hold until the previous task finishes. Thus the execution time of the tasks in the queue increases. In this section, we propose a Task Scheduling algorithm to manage the task assignment between the managers and servies.

There are two kinds of task scheduling: static and dynamic. We employed dynamic task scheduling in our solution. Basically, each service just finish an atomic task, so it is easy to estimate the influence caused by work load. In Algorithm 1, we show an example which updates the execution time of the service dynamically. *EstimatedExecutionTime* is the estimation of the service execution time for each task. When receiving a task, the increment of *CurrentExecutionTime* is *EstimatedExecutionTime*. When finishing a task, *CurrentExecutionTime* will reduce *EstimatedExecutionTime*.

However, the overhead of the approach proposed above is very high. The accomplishment of a task requires two messages, one for the task request and another for the task

Algorithm 3.1 Task Scheduling Algorithm

EstimatedExecutionTime: the estimated execution time of one unit task.

CurrentExecutionTime: current accumulated execution time of one service.

task: the task in the service, which is assigned by the service manager.

BEGIN

if(Receive task){

CurrentExecutionTime+=*EstimatedExecutionTime*;

addTaskQueue(task);

updateQoS();

}

if(Finish task){

CurrentExecutionTime-=*EstimatedExecutionTime*;

removeTaskQueue(task);

updateQoS();

}

END

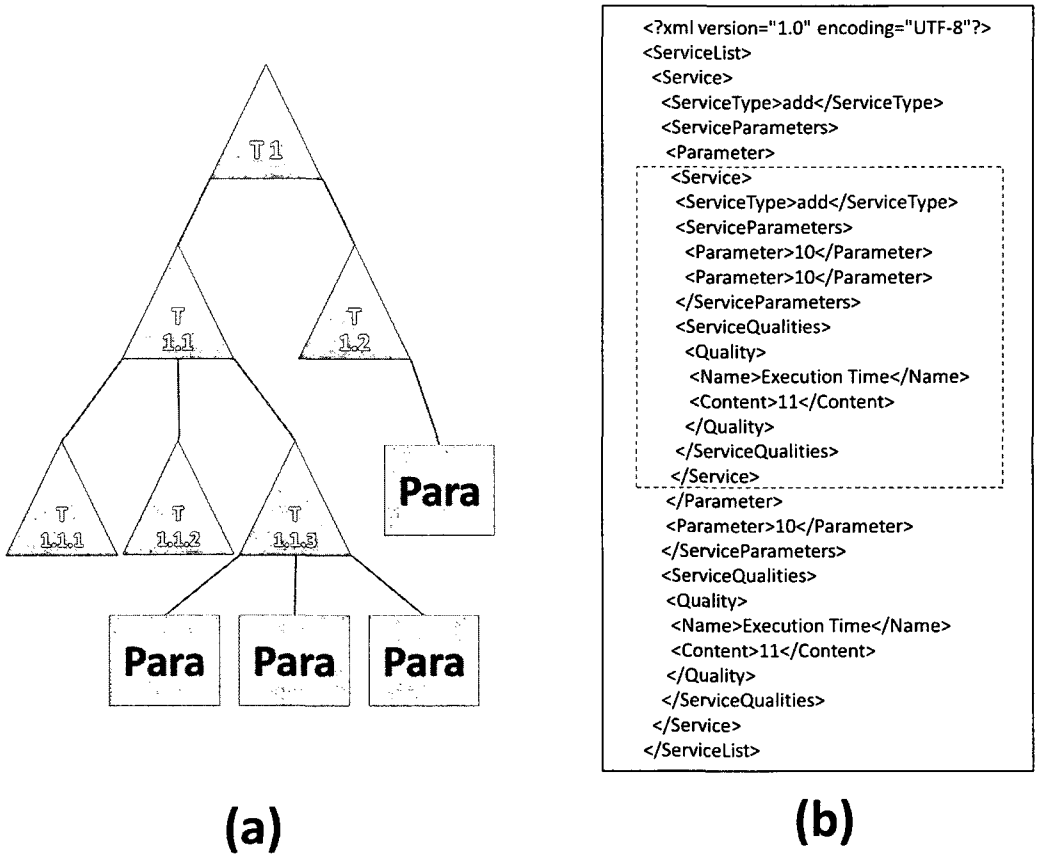
result. As we see in Algorithm 1, when a service receives or finishes a task, it updates its QoS values to the services manager. This process doubles the number of messages for each task. To improve this situation, we will not allow the service to directly update *CurrentExecutionTime* to the service manager. The service manager records the *EstimatedExecutionTime* of the service and increase the service's *CurrentExecutionTime* when it assigns a task to this service. When a service finishes a task, it needs to send a small "Done" message to its manager with its name in the message.

Based on our task scheduling algorithm, tasks will be equally distributed to proper services. If each service performs according to our assumption, the QoS success rate will be very high. However, frequently there are many services run on each server node. They share the memory and CPU of the node and need to wait in queue and access these resources one by one. If a server is overloaded, it will increase the overhead of tasks and lower the success rate. In order to guarantee the success rate, we will migrate some services from the overloaded server to a lightly loaded server. Our first step would be to monitor the work load on each server, which will be discussed in the next section.

3.2 Task Specification and Service Composition

In our proposed system, a task is described as an XML document as shown in Figure 3.3. This task description sets up the functional requirements of the task, specifies the task's internal dependency and also defines the QoS parameters. For example, the Service tag can be embedded in the Parameter tag, specifying that other services' return data can be used as input data for the current service. Within the Service tag, service quality requirements can be added within the ServiceQualities tag. The task XML can be easily expanded to satisfy any complex task description with full QoS-awareness.

This task structure is easy to understand and reorganized. Each service is able to accept several parameters and to generate one output. Parameters can be data or output from other services. With this structure, we can create a task tree; this can describe




```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceList>
 <Service>
  <ServiceType>add</ServiceType>
  <ServiceParameters>
   <Parameter>
    <Service>
     <ServiceType>add</ServiceType>
     <ServiceParameters>
      <Parameter>10</Parameter>
      <Parameter>10</Parameter>
     </ServiceParameters>
     <ServiceQualities>
      <Quality>
       <Name>Execution Time</Name>
       <Content>11</Content>
      </Quality>
     </ServiceQualities>
    </Service>
   </Parameter>
   <Parameter>10</Parameter>
  </ServiceParameters>
  <ServiceQualities>
   <Quality>
    <Name>Execution Time</Name>
    <Content>11</Content>
   </Quality>
  </ServiceQualities>
 </Service>
</ServiceList>
```

Figure 3.3: (a) Example of a task specification structure (b) Example of a task specification in XML [45]

tasks in XML as shown in Figure 3.3(b). The output of the root is the final result for the request. The purpose of designing this kind of task structure is to adjust the task structure to our system structure. In our system, a service manager examines part of the task and forwards the rest of the task to the children service managers split the workload. With this task structure, it's easy to implement such a feature. As shown in Figure 3.3(a), in the example of a task specification structure, a main service manager receives the task and examines the task from the root to the leaves. The main service manager then takes a part of the task that it can finish with the services that it owns. For example, a main service manager can finish T1, but not T1.1 and T1.2. Thus the main service manager takes T1 from the task and split the task into three parts, the T1, sub-tree root at T1.1 and the sub-tree root at T1.2. The two sub-trees, which remain in task structure, can be treated as two new tasks and forwarded to the children service managers. Thus, these two sub-trees can be examined at the same time by different branches of the main service manager. This not only splits the analysis workload among components, but also enables the task analysis to be executed in parallel processes. To consist with JXTA, we implemented our task specification in XML. Another reason for the implementation of our task specification in XML is that the XML document's format is very closed to our design. It is therefore simple to implement our design in an XML document.

Furthermore, in our QoS-aware service composition and management framework, services are defined as basic functional components that execute a task or task components. Before services can be used in the system, they must register themselves to the service manager. The service registration request is also implemented in XML format as shown in Figure 3.4. In this XML registration document, the parameters processed by this service are defined, as well as the name and the parameters' data type. In our system, services cannot be directly used by the application tasks. Each service is managed by one of the service managers, and has its own QoS-serving capability. When a service serves a task or task components, its QoS capability may decrease. Our system therefore

provides a mechanism that allows the service to update its service manager dynamically.

```
<?xml version="1.0" encoding="ISO-8859-
1" ?>
<ServiceRegister>
  <ServiceType>add</ServiceType>
  <ServiceName>add1</ServiceName>
   <ServiceParameters>
    <Parameter>
     <Name>para1</Name>
     <Type>integer</Type>
    </Parameter>
    <Parameter>
     <Name>para2</Name>
     <Type>integer</Type>
    </Parameter>
   </ServiceParameters>
  <ServiceQualities>
   <Quality>
    <Name>Execution Time</Name>
    <Content>250</Content>
   </Quality>
   <Quality>
    <Name>faulty</Name>
    <Content>10</Content>
   </Quality>
 </ServiceQualities>
</ServiceRegister>
```

Figure 3.4: Example of Service Registration Document[45]

Chapter 4

Load Balancing Scheme

Quality of Service (QoS) has achieved a great deal of popularity in many areas, such as entertainment services, emergency services and transaction services. However, due to extreme increases in requests and limited resources, load balancing is one of the most important topics of QoS. Most load balancing algorithms on QoS-aware systems focus on Routing Mechanisms and Traffic Engineering. There is limited research on Task Scheduling and Service Migration. We first propose a Task Scheduling by dynamic QoS properties in Section 3.1.1. In this chapter, in addition to proposing a Services Migration by Genetic Algorithm to optimize the performance of our system. We implemented a prototype of our algorithm on our P2P QoS-aware system with JXTA techniques and perform a simulation test to analyze our solution. We compared our algorithm with two other systems, a system without any load balance and a system that only using the task scheduling algorithm. The experiment results demonstrated that our solution is superior to the other two and that it leads to a higher success rate for QoS.

4.1 Load Balancing Scheme

We first propose a Task Scheduling solution on Dynamic QoS Properties for the Load Balancing Scheme in Section 3.1.1. Furthermore, we propose an additional Service Mi-

gration on the Genetic Algorithm in the scheme to optimize the performance of the system. We implemented this scheme on our previous QoS-Aware system framework in Figure 3.1 with a JXTA technique.

4.1.1 Work Load Monitor

Information, such as CPU and memory usage, needs to be measured for load balancing. However, presented below are several reasons why these properties are not measured: Different systems have their own measurement commands for system performance, which are not good for a heterogeneous network. Moreover, some development platforms, like JAVA, are difficult to implement on, and system call is very expensive. Therefore, we tried to discover other properties that can be measured easily and efficiently.

As we discussed in the section above, if a server is overloaded, the QoS success rate will decrease. Thus monitoring the QoS success rate is another way to monitor the work load of servers. In our design, each server node measures its success rate by collecting the QoS information from all the services that run on it. Server nodes will receive the parameters, *SuccessRate* and *TotalTasksNumber,* of all the services.

$$SuccessRate_{node} = \frac{\sum(SuccessRate_i \times TotalTasksNumber_i)}{\sum TotalTasksNumber_i} \quad (4.1)$$

Where:

SuccessRate is the success rate of user requests in a current node or service

$TotalTasksNumber$ is the total number of tasks in one service

i is the number of service

We set *RequiredSuccessRate* as property to measure if a service is overloaded or not. After a server calculates its success rate, it compares $SuccessRate_{node}$ to *RequiredSuccessRate.* If it's overloaded, it sends an "Overload" message to its service manager. After which, the service manager decides whether this server needs to migrate some services to other servers and which services will be moved.

4.1.2 Migration Decision

Even if a service manager receives an "Overload" message from a monitored node, it may be useless to move the services. If all nodes in the system are in an overloaded condition, there are too many tasks received from users and not enough resources to guarantee the quality of services. If service managers move some services in this situation, they will use the limited resources and decrease the success rate of QoS. Thus, before a Load Balance Manager decides to move a service, it needs to understand the workload of each node in the network which it monitors and manages. If all nodes are overloaded, the Load Balance Manager will inform all the nodes and decrease their *RequiredSuccessRate* properties. In this situation, the whole system will achieve a low performance level.

On the other hand, a server node is still in recovery, after finishing a service migration. It requires time to raise the success rate, and its success rate is lower than *RequiredSuccessRate*. The monitor will send an "Overload" message again, which is unnecessary and damaged. The server will continue moving services and will never recover. To solve this problem, the *RequiredSuccessRate* must be decreased, when a server finishes services reconfiguration. It also needs to minus an α which is caused by the overhead of migration. The overhead of migration adds to the tasks in the migrated service. This overhead is caused by many factors, like the size of services, network speed and the number of services that need to be migrated. When a service is being moved, it will occupy some bandwidth and the resources of the system will decrease. All these factors lead to the decrement of the success rate. Based in this experiment, the success rate averagely decreases by about 5% for each time the system performs a reconfiguration in our prototype.

$$RequiredSuccessRate_{node} = SuccessRate_{node} - \alpha \quad (4.2)$$

Where:

RequiredSuccessRate is the required success rate of user requests for current

$0 < \alpha < 1$

α is the estimated overhead caused by service migration, it is 5% in our experiment

4.1.3 Services Migration Based On Genetic Algorithm

The following situation will illustrate the necessity for services migration in QoS-aware systems: all services completing the same function are located in one server, and concurrently all the requests desire access to these services. In this case, all requests rush into this server. This scenario illustrates why we need services migration in QoS-aware systems is needed.

4.1.3.1 Load Balance Algorithm

As one kind of evolutionary algorithm, the Genetic Algorithm does a good job finding the approximate solution of a searching problem. Some researchers use Genetic Algorithm as a service composition algorithm on QoS-aware to find an appropriate service path, which has been done in [8, 7]. Differently, we use Genetic Algorithm to find an appropriate system structure which will maximize the usage of available resources. In this section, we propose a service load balance solution based on the Genetic Algorithm. There are three steps in our solution: 1) To obtain a table which represents current system structure; 2) the Load Balance Manager uses the Genetic Algorithm to generate enough generation and chooses the one which is nearly optimized; 3) The Load Balance Manager move services one by one based on the system structure blueprint which is selected. There are three genetic operators in Genetic Algorithm: crossover, mutation and selection.

Algorithm 2 shows the procedure to generate the image of the system structure which is the nearest optimal closet .

In our solution, we assume that if one service has a large *TotalTasksNumber* it will be assigned more tasks later. The reason for this assumption is that if a service has a lot of tasks, most tasks have recently accessed this service and found it to have a good quality. Thus, the probability for assigning tasks to this service is very high.

Crossover:

Crossover is the first step in our solution to generate a child system structure from the initial population. It tries to balance the work load with the worst node and the best one. In the crossover step, we chose the simple two-point crossover, which exchanges two services on the light load server and heavy workload server. It chooses the service with the smallest *TotalTasksNumber* in the lowest *SuccessRate* server node, and switches it with the service with the biggest *TotalTasksNumber* in the highest *SuccessRate* server node.

Mutation:

Mutation generates a set of possible children which are slightly different. The input of mutation is the child that is generated in the crossover step. It chooses one service randomly among all the services and moves it to a random server node other than the one it stays. It keeps doing this repeatedly until enough children have been generated.

Selection:

The selection function chooses one or some individuals from the population generated from crossover and mutation steps. The selected individuals are collected together to create a new initial population for the next loop. We use Deterministic Tournament Selection in our algorithm, so there is only one winner at the end. In the selection step, the $SuccessRate_{node}$ of all nodes is recalculated. Assuming n is the number of all nodes, the selection function compares the workload difference of all the nodes in the system as follow:

1) If n is even, create i=$n/2$:

Algorithm 4.1 Algorithm for Generating Services Migration Blueprint

ϕ: a set of populations

P: The initial population with a set of individuals.

P_{new}: a new set of individuals generated by Genetic Algorithm

S_i: current individual

S_{best}: the best individual among the population

BEGIN

while(Not meet termination situation){

// Crossover step

Select one service from the light workload node;

Select one service from the heavy workload node;

Swap these two services;

Create new population P_{new} to ϕ;

while(Not enough different individual S){

// Mutation step

Move one service randomly to create a new structure S_i;

S_i add to P_{new};

}

// Selection step

Select the best system structure S_{best} from P_{new};

Clear P;

S_{best} add to P ;

Set P to ϕ ;

}

END

$$Difference_{workload} = \sum |Workload_{2i} - Workload_{2i+1}|$$

2) If n is odd, create i=$\lfloor n/2 \rfloor$:

$$Difference_{workload} = \sum |Workload_{2i} - Workload_{2i+1}| + |Workload_n - AverageWorkload|$$

Where:

i is number of current service

n is the total number of services

Workload is the workload of a node

Difference is the difference of the workload among all nodes

The system structure with the smallest $Difference_{workload}$ is the most balanced. It will be chosen as the input for the next round or as the final table. The Load Balance Manager will move the service based on this table.

4.1.3.2 Services Migration

After selecting the final result of the optimized system structure, the Load Balance Manager start moving the services to the destination decided in the final table. To avoid using too much bandwidth for services migration, only one service is allowed to move at one time and the other services work as usual.

To move the service, the Load Balance Manager freezes the chosen service in the registration list; this ensure that no tasks is sent to this service until it finishes the migration. We need to find a way to save states and the task list of this service. The JAVA interface *Serializable* is used to save the service object into byte array. The serialized services are then sent to the destination server nodes and the services are uploaded into memory. Because we use JXTA as transportation infrastructure and Propagate Pipe in JXTA as our main transportation way, no information in the manager side requires to be changed. After service is uploaded, it sends a "WakeUp" message to the Load Balance Manager. The Load Balance Manager simply changes the state of the service in the registration list and everything works the same as before migration.

On the Service Platform there are two components: Migration Service and Upload Service; these are used for services migration. The procedure of service migration is shown in Figure 4.1. First, the Load Balance Manager sends a "Migration" message to the service that need to be moved. The chosen service informs the original Service Platform that it is located in. After getting the confirmation from the original Service Platform, the service freezes and serializes itself. The Service Platform also inform the Migration Service to prepare the migration. The Migration Service builds up connection to the Upload Service of destination Service Platform. After the above-mentioned steps, Migration Service receives the serialized object and sends it to the Upload Service of the target Service Platform. The Upload Service loads the serialized object into memory and activate the object.

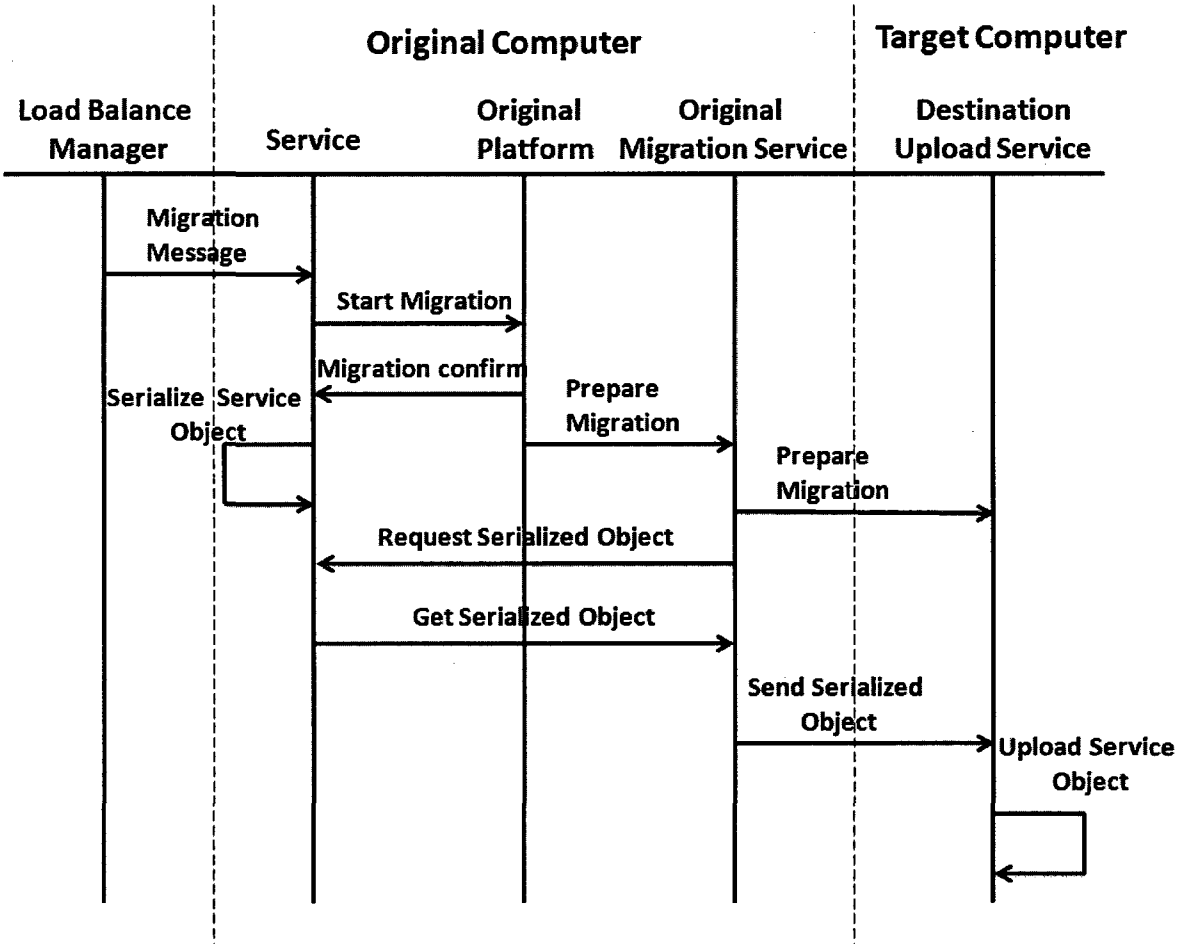


Figure 4.1: Procedure for Service Migration

Chapter 5

Design of a SOA-based System

Service Oriented Architecture (SOA) [14] rapidly became a dominant approach for system development and integration, which allows different software service components to exchange information through various networks. SOA is based on the concept of "services" defined as software units that use predefined protocols to communicate with each other. In service-oriented systems, services expose themselves by using software interfaces; service consumers or users can then easily access services through these interfaces without an awareness of the detailed service implementation. The main advantages of SOA are flexibility, interoperability and reusability. Indeed, SOA is a promising technique, which has attracted many businesses to implement their key software backbones using SOA architecture. For instance, most online businesses currently are implemented using SOA, and many companies run their business-to-business applications based on SOA backend.

As a matter of fact, more and more researchers have dedicated themselves to discover the new features of SOA. However, a major concern is to effectively design a service-oriented system considering its design complexity. With the increasing usage of SOA, necessary services are also increasing rapidly. Some of these services may be composed of thousands of smaller components, which may overwhelm system designers. On the other hand, in order to satisfy the requirements of security and authorization, more func-

tionality needs to be added to original services. Recent business processes are becoming more and more complex; this requires designers to greatly consider the initialization and maintenance of the whole system's interoperations.

To solve the aforementioned concerns, we propose a novel modeling tool, JXTA Message Layer, targeted at solving SOA related system design problems. We implemented this tool based on Sun's JXTA P2P framework. It is worth mentioning that our tool itself is built on JXTA's service backend; thus, it easily and accurately models many real world and real-time SOA based systems, especially P2P based system architectures. Our framework is based on a message layer and P2P based services, which is able to model SOA systems in a real-time fashion. Because our tool is built upon existing JXTA API, the services in our framework can easily communicate with each other using JXTA pipes: these can be either unicast or propagate pipes. Moreover, a set of services can be organized as a group in JXTA, which reduces the complexity of the interoperation of services. With the discovery protocol of JXTA, services can publish or discover advertisements efficiently across a network. Furthermore, different subnets can communicate with each other using super, rendezvous and relay peers. With these features of JXTA, it's simple to model a service-oriented system using our tool.

5.1 Framework Design

In this chapter, we present our design, and the means by which we implemented, a message-oriented and real-time SOA based modeling framework on top of JXTA P2P protocols. In our framework, all services and components are implemented as JXTA entities, which create JXTA pipes to communicate with each other. Services can publish advertisements on the network; these can be discovered by other services easily through JXTA discovery protocol. For JXTA entities residing on different networks, JXTA provides super peers, rendezvous and relay peers to support inter-subnet message communication. In a normal JXTA network, entities initialize connections with other

entities after discovering their advertisements. However, each JXTA pipe is maintained as a Java object in the memory. When the number of JXTA entities and connections increase, the usage of memory dramatically increase simultaneously.

To solve this problem, we created a Message Layer in our framework, as shown in Figure 5.1. This message layer is responsible for managing the whole message exchanging process, including pipe creation, and the sending and receiving of messages. In particular, our approach is to separate the process of an advertisement's discovery from pipe initialization. In the pipe initialization process, entities do not create any real pipes. If an entity needs to send a message, it first discovers the advertisement of its destination, and then sends the message to the Message Layer. When the Message Layer receives any messages, it redirects them to the destination with the information included in the message. In other words, all messages in our framework are exchanged through the Message Layer. We also have implemented a Component Platform, which works as a monitoring and organizing manager for each node.

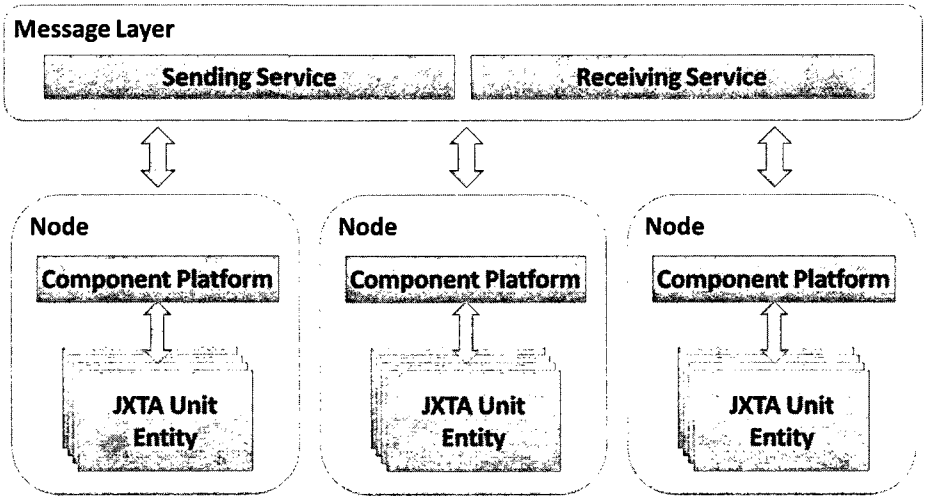


Figure 5.1: Framework design for SOA on JXTA

The advantages of this system structure are: 1) It reduces the number of pipe objects that need to be maintained. Sometimes parts of pipe objects may not be used frequently, but they still exist in memory, which results in a waste of system resources. Thus, by

decreasing the number of pipe objects it can save the system's resources; 2) In our approach, we simplify the process for component migration when necessary. In SOA systems, components may need to be moved from one node to another node during a load balancing or fault tolerant process. In our solution, components are not required to be aware of other components' movement. All components can still work as before after the migration and they do not need to establish the connection again; and 3) We also simplify the implementation of reliable JXTA pipes. In our solution, we can implement the reliable features of JXTA pipes on the Message Layer, without implementing them on all the services.

5.2 System Implementation

In our framework, we try to efficiently utilize existing features of JXTA, such as the concept of peer group and peer advertisement, and pipe and service discovery protocols. All these features can make applications more flexible and configurable. However, improvements still have to be done in order to implement our design presented in the previous section.

As we know, the JXTA P2P network is a virtual overlay network. To implement or simulate a SOA system on a JXTA overlay network, an application needs to first start a JXTA virtual network. It also needs to create a global netPeerGroup object for each application. Sometimes, there exist a set of services running on one node, and they do not modify the netPeerGroup object. Usually, these services create a new peer group under the netPeerGroup object. Our idea is to share one netPeerGroup object among all the application on the same node; the Component Platform's implementation is undergone to fulfill this idea. Moreover, in our approach, the JXTA entities on the same node are implemented as threads under the Component Platform; consequently the JXTA entities all shares the netPeerGroup object. Another advantage of our scheme is that all JXTA entities share the same cache folder, which is used to save the discovered

advertisements for future usage. Advertisements found by JXTA entity are saved in the cache folder. Due to the cache folder's sharing mechanism, other services can access this same advertisement and therefore save time that would otherwise be used for searching advertisements.

Our plan encounters another problem in the Message Layer's tendency to double messages that needs to be sent. In our design, each unit-entity must send message to Message Layer first. The Message Layer then redirects these messages to the specified unit entity. This procedure makes each message into two messages. The initial message is sent from the unit entity to the Message Layer, and the other passes from the Message Layer to the unit entity. To solve this problem, we develop the Message Layer into a co-operated Message Layer on each node, as shown on Figure 5.2. In such an approach, there is a Message Layer running on each node, and all Message Layer components connect to each other to form a larger Message Layer across the network. JXTA entities work the same way as we described previously in section 5.1. However, JXTA entities residing in the same node do not send messages to the Message Layer through JXTA pipes. Instead, they use method call to send message to the Message Layer which is much faster. In other words, if two entities are running on the same node, their communication is completely manipulated by method call without using any JXTA message. Thus, by using the co-operated Message Layer, we significantly reduce potential communication overhead among JXTA entities.

In order to separate the discovery process from the pipe initialization process, we have also improved upon the discovery process. The real JXTA pipes only exist among Message Layers, so we do not need to create any JXTA pipes in JXTA entities. However, the service advertisement of JXTA needs to contain pipe advertisement for other entities to create the connection. In our framework, each entity no longer contain any pipe information. We still want to use the discovery process that JXTA supplies, therefore, improvements are required when encapsulating the advertisement. When creating the service advertisement, each entity will inform the Message Layer that it belongs to. The

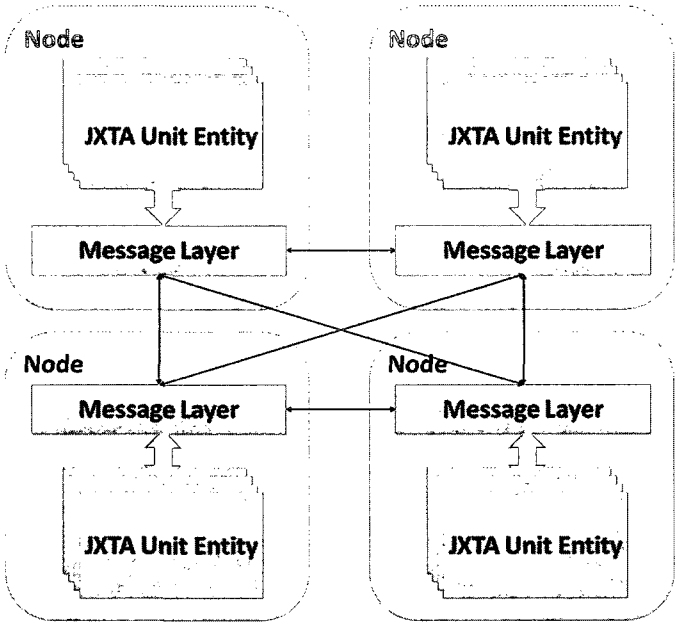


Figure 5.2: Improved cooperated Message Layer structure

Message Layer then creates an input pipe based upon the pipe advertisement of the entity that publishes the service advertisement. Though the advertisement is published by entities, the pipe is created in the Message Layer. For example, when entity A discovers the service advertisement of entity B, it can retrieve the pipe advertisement of entity B from the service advertisement. Entity A will create an output pipe based on this pipe advertisement. However, the output pipe, that entity A creates, is connected to the Message Layer, not entity B. Afterwards, entity A sends the pipe's information to its Message Layer to create a connection to entity B's Message Layer. Now entity A is able to send messages to entity B.

It is important to note mentioning that our solution does not affect other features of the JXTA network. Users are still able to create peer group to confine the communication between entities and to organize overall the system structure. Although all Message Layers are on the same peer group (there are no communication limit on the Message Layer), the process of advertisement discovery is still under control by peer group policy. This is another reason why we want to separate the communication process from the

advertisement discovery process. We will provide an example of a scenario to illustrate this matter. Entity A in Group 1 desires to communicate with Entity B in Group 2. Technically, Entity A is able to connect to Entity B through the Message Layer. However, Entity A will never find the service advertisement of Entity B because the discovery protocol of JXTA prevents this situation.

Chapter 6

Implementation of Tree-based Algorithm

In order to verify our design of the Tree-based QoS-aware framework, we implemented and tested the system on the JXTA Message Layer proposed in Chapter 5. In this chapter, we will present the detail about the implementation and experiments of our design.

6.1 Implementation

We implemented our QoS-aware system on the SOA framework proposed on the JXTA Message Layer. The architecture of the QoS-aware system is shown in Figure 3.1. A comparison of our design with the flat-based service structure is shown in Figure 3.2, which is widely used in service composition systems. We implemented and tested both structures using our SOA based framework to compare the performance of these two systems.

In our design, there are two main components in the system, manager and service. Both of them are implemented as JXTA entities. The basic functions of manager components are to manage the services, to assign tasks and to evaluate the performance of the

services. The Main Service Manager is the main manager that both receives tasks from the user and also determines the services a task should be assigned to. Other Service Managers are controlled by managers on the upper level. In this system, services are the unit functions of the system; each service can finish one kind of unit tasks. Service Mangers need to compose a set of services to complete a certain task submitted by a user. Each Service Manager knows only the components directly connected to it. Upon reception of a task from a user, the Main Service Manager forwards it to the Service Managers controlled by the Main Service Manager. The Service Manager that receives this task from a higher level manager will propose a service composition plan for part of the task. It will then eliminate that part of the task and send the newly defined task to the Service Manage it controls on the lower level that it controls. After the whole task has been assigned, the system will begin the execution process from the bottom to the top; the final result is generated on the level of the Main Service Manager. As a matter of fact, the Service Managers that do not depend on each other work concurrently. In contrast, in the flat-based service structure, only one Main Service Manager handles the service composition as well as the monitoring of the underlying services.

It is straightforward to compare the two systems' architectures using our framework as shown in Figure 6.1. On the virtual communication layer that is built by the JXTA Message Layer, each component is able to connect to components it discovers. Components do not need to know the node on which the destination component runs. We can see from Figure 6.1 that the Main Service Manager runs on a separated node, because it needs more resources to process tasks. In a cluster of computers, we can run the service managers and services in whatever fashion we want. Our real-time experiments demonstrated that our proposed architecture, shown in Figure 3.1, has a better performance than the architecture in Figure 3.2.

In the task scheduling algorithm that we proposed in Section 3.1.1, we created some dynamic QoS properties for monitoring and estimating the performance of the services. Based on these QoS properties, the service manager makes a decision concerning task

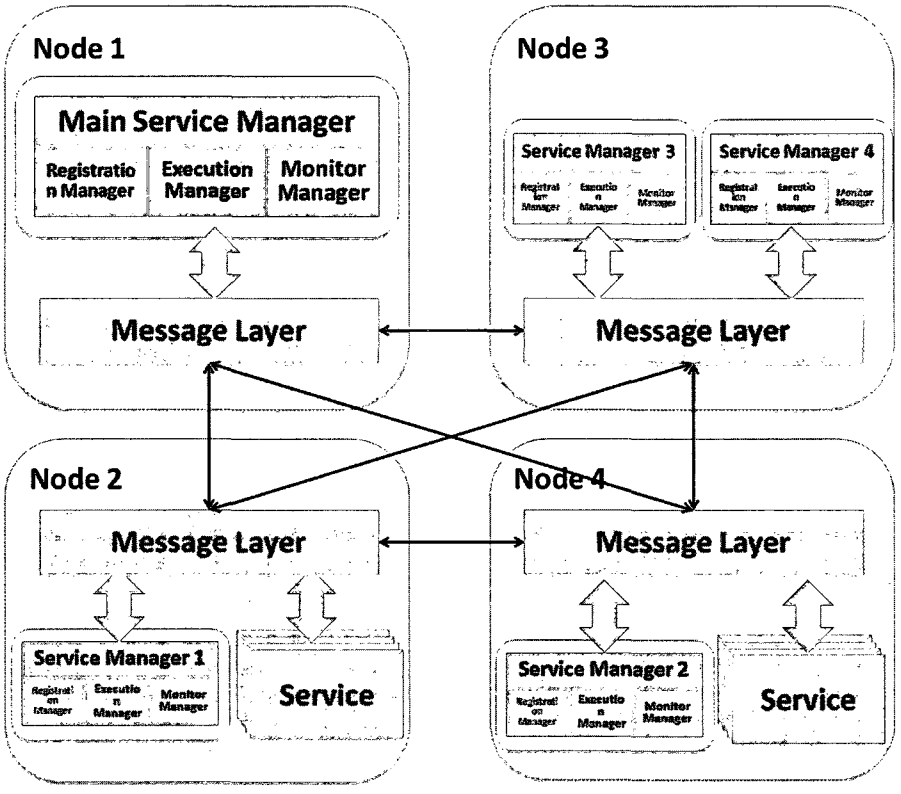


Figure 6.1: Implementation of QoS-aware system on the SOA framework

assignment. In the service migration algorithm, there is a load balancing manager monitoring the whole system that it controls. It probes every service and gets the big picture of the whole system. Then, it uses the Genetic Algorithm to generate one of the best arrangements for the system. After getting the final blueprint of the system, the service manager starts the service migration in order to get the final arrangement created by the Genetic Algorithm. It is worth noticing that we also implemented a user monitoring service in our JXTA Message Layer framework for monitoring the system across the subnets.

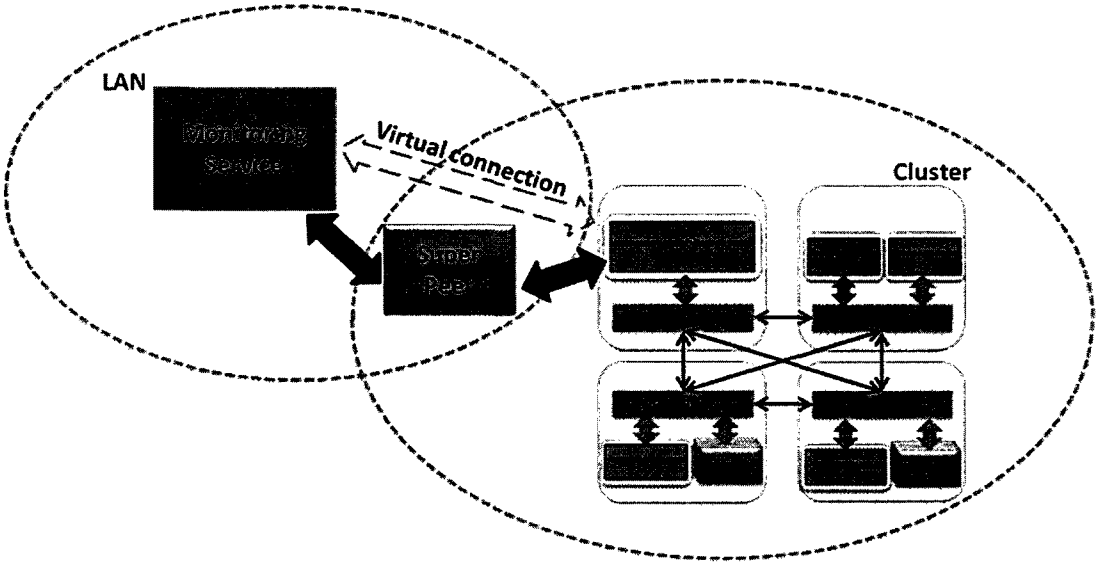


Figure 6.2: A super peer creates a virtual connection between two subnets

The system structure is almost the same as Figure 6.1. However, we implemented some additional components in order to add the new scheduling function and service migration function. Due to the unique features of our framework, the service manager and services do not need to establish any connection to the migration service. The migration service only needs to communicate with the service manager when it starts and finishes migration. We also evaluated the overhead of the service migration through a experiment in our framework, and found that the overhead is at an acceptable level. In fact, the services built on our framework migrate easily and efficiently from one node

to another. This is very useful for the implementation of various service composition schema and load-balancing algorithms.

Moreover, we also created some JXTA super peers to support the communication across the network. As shown in Figure 6.2, a super peer runs on both sub-networks that need to communicate with each other. The involved JXTA entities register to the super peer to publish and discover the advertisements to other subnets. We have also conducted some experiments based on such a scenario.

6.2 Experiments

6.2.1 System Architecture Experiments

In the following experiment, we compare our tree-based service composition and management framework with the flat-based service composition and management framework, in terms of system scalability with regard to the increased number of service components. This experiment was conducted under a local network in which multiple desktop computers were interconnected through 100M Ethernet. Each desktop computer had a 3.0GHz CPU of Intel Pentium 4 with 512M memory, and was installed with Fedora 5, JDK 1.6.04 and JXTA 2.5.

In this experiment, we used two types of services, each of which had three different kinds of QoS-serving parameters. Therefore, we had six different services which were then controlled by six service managers in our tree-based design. In contrast, all the services were connected to one Main Service Manager in the flat-based service composition and management framework. We used one Java process to generate tasks, which are then sent to the system using predefined sending rates. The Main Service Manager and other lower-level managers used the "First Come First Serve" rule to handle incoming tasks. Moreover, all managers and services have task queues that store incoming tasks before processing. This experiment was run on 4 desktop computers with a fixed 400ms task-sending interval. We measured how task success rates change with increased number of

Figure 6.3: Result of Measuring Scalability[45]

service components. The experiment result is shown in Figure 6.3, where the tree-based service framework is shown to have significantly higher success rates than flat-based service framework. Moreover, the tree-based service framework maintains a high success rate with an increased number of service components in the system. This means it has better scalability in terms of the number of service components changes. We believe there are two main reasons for this difference. In a flat-based service framework, the Main Service Manager becomes the bottleneck of the whole system. When the number of service components is increased, the Main Service Manager's workload increases as well; it needs to handle a larger service list when analyzing the task. Secondly, in the tree-based framework, Main Service Manager's workload is reduced, and multiple tasks can be handled concurrently on different children managers on different machines.

Furthermore, the performance of both frameworks dropped to 20% when the number of services was increased from 40 to 50. We suspected this situation to have been caused by the limitation of computing resources. We therefore added one more desktop and

found that the success rate of the tree-based framework increased to 80%. Conversely, the success rate of the flat-based service framework was still between 40%-50%.

Another experiment compared the task operation capability of these two frameworks. We used 4 desktops to run this experiment with a 3 level task tree. We also calculated the success rate of the two frameworks. However, we decreased the task sending interval from 500ms to 100ms. We chose 10 as the number of service; our information from the first experiment demonstrates that that when there are 10 services the performance of these two frameworks is almost the same. As shown in fig.6.4, the services performance is initially very similar. However after the sending interval lowers more than 400ms, the flat-based service framework decreases much more quickly than that of the tree-based framework. The execution of a flat-based service framework is sequential; therefore, the delay of previous tasks will affect following tasks. If the number of task increase, the situation will be worse. Moreover, tree-based service frameworks can work congruently. This can efficiently reduce the effect of delay caused by previous tasks.

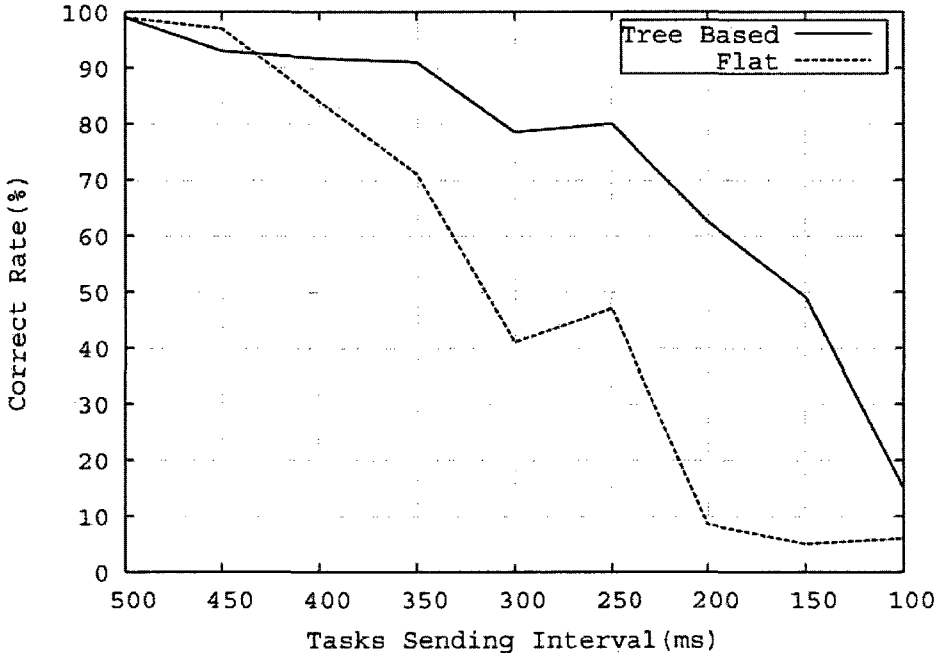


Figure 6.4: Result of Measuring Task Operation Capability

6.2.2 Network Emulation Experiment

We implemented a prototype of our load balancing algorithm and tested it on our P2P QoS-aware system. We tested our prototype with three computers. The configuration of each computer is shown in Table 6.1.

Device	Quality
CPU	Intel Pentium 4 3.40GHz
Memory	512MB

Table 6.1: Environment of Experiment

To optimize the performance of JXTA, we designed a services structure in each server node in Chapter 5, shown in Figure 6.5. The Service Platform is a process running on the server. Servers are threads under the Service Platform. In this structure, there is only one NetPeerGroup entity and only one cache folder on the server; both are created by the Service Platform. The Service Platform shares the NetPeerGroup entity to all services. All services save their own routing information in the same cache folder. If one service needs to communicate to another service in the same server node, it can easily get the routing information from the cache without requesting it from the routing peer.

The architecture of the system for testing the prototype of our load balancing scheme is shown in Figure 6.6. The Manager is running on one computer. It contains the following three components: Registration Manager, Assignment Manager and Load Balance Manager. Services are running on two computers. On each computer, services are integrated as the service structure proposed above.

6.2.2.1 Experiment Result

We compared our prototype with two other models. System 1 is our prototype. System 2 is the service system that only has Task Scheduling. System 3 is the service system

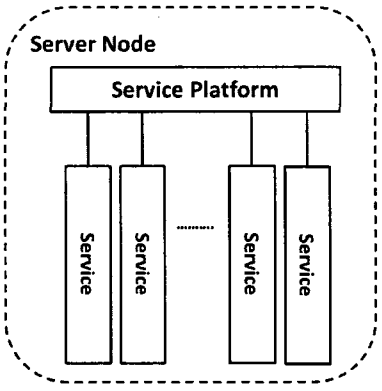


Figure 6.5: Service Structure in Server Node

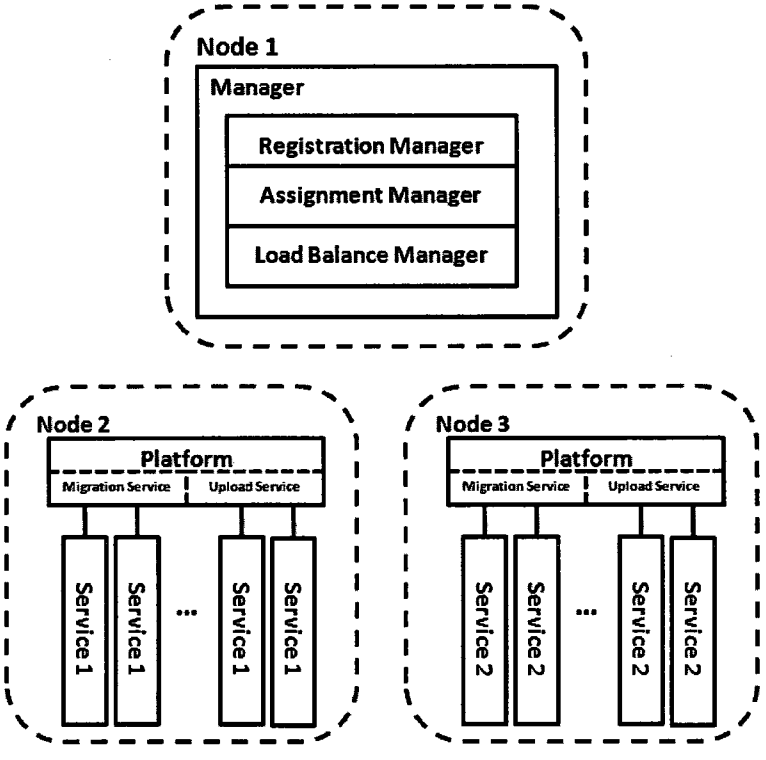


Figure 6.6: Architecture of Genetic Algorithm Load Balance Prototype

without any Load Balancing Algorithms. In all these systems, the structure is almost the same as Figure 6.6. However, System 2 and System 3 do not have the Load Balance Manager. In System 3, the Services do not update QoS properties to the Manager. The QoS property measured in our experiment is Execution Time, which is very sensitive to workload. We tested the total QoS success rate of the whole system among these three systems and compared them together.

In the first experiment, we sent the tasks in fixed intervals of 100ms. As shown in Figure 6.6, each computer has different types of services that finish different work. Each computer has 20 services. However, tasks mainly access one kind of service. Therefore task distribution is imbalanced between two computers. In this structure we test how well our proposed service migration works. We set the *RequiredSuccessRate* of our prototype to 85%. When the success rate is lower than 85%, it reorganizes the services.

The result of the experiment is shown in Figure 6.7. From this result we can determine that our prototype works much better than the other two. Because System 3 has no load balance algorithms, some services that have good qualities are chosen frequently. As a result, the success rate of execution time becomes extremely low. System 2 is obviously better than System 3, because it assigned tasks to suitable services equally. However, the main reason for the failure of the QoS is that most tasks rush to one server. At the beginning, the success rate of System 1, which uses our proposed Genetic Algorithm Load Balancing shceme, goes down. However, it quickly returns to a normal level. The success rate decreases and activates the Load Balance Manager which reorganizes the system and the workload becomes balanced in the whole system. From those result, we also can determine that the overhead caused by service migration is not very high, because the success rate returns to a normal level very soon.

In the second experiment, we tested the three systems with a different task sending rate. We measured the capability of these three systems in this experiment, and tested if the system can endure the large amount of task that concurrently rush into the system. We tested the systems with 3000 tasks. The interval time of a task was from 100ms to

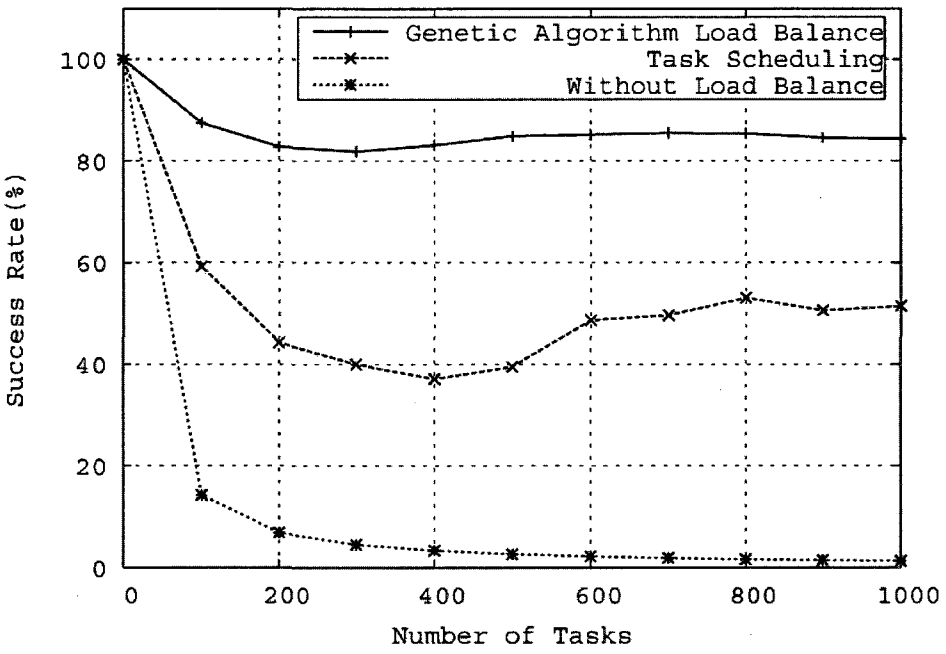


Figure 6.7: First Comparison Experiment Result of Three Systems

10ms. The result is shown in Figure 6.8. As we can see from Figure 6.8, the success rate of System 3 is still extremely low. Our proposed System 1 works much better than System 2. In Figure 6.8, over a period of time, System 1 can supply performance on a stable level, while the success rate of System 2 continues to go down. The performance of System 3 shows that it is more efficient with tolerating large amount of task requests.

6.2.2.2 Simulation Experiment

We conducted a simulation analysis to test our system. Based on this analysis, we will improve the development of our load balancing algorithm in the future. The simulation environment is shown in Figure 6.9. The structure of the system is the same with the emulation experiment in the above-mentioned section. All entities are controlled by the Time Manager, which controls the time advance.

Like the first experiment, we measured the success rate of these three systems. We performed the simulation on one computer with a multi-thread technique. The Time

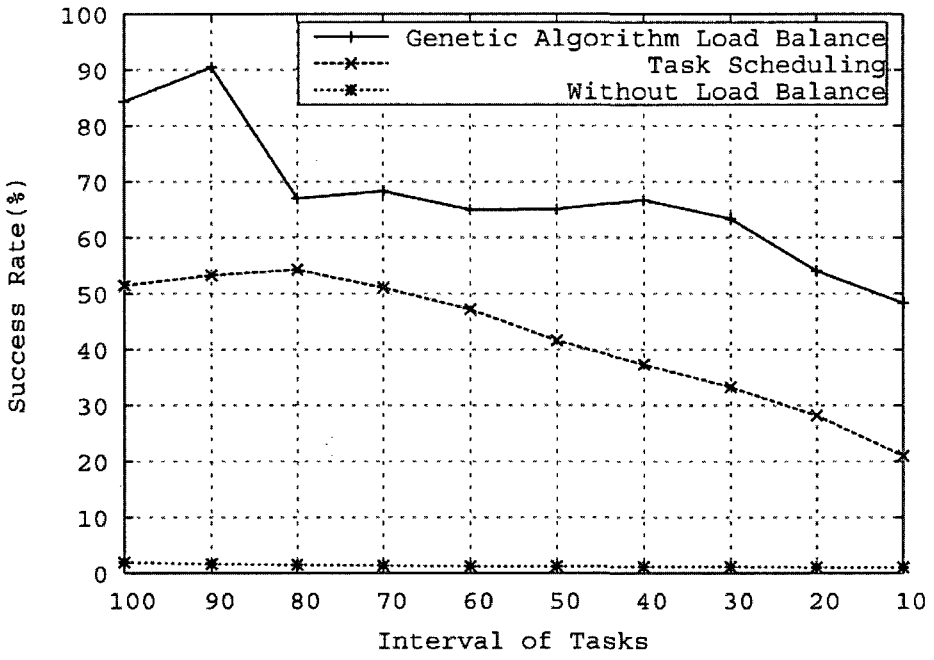


Figure 6.8: First Comparison Experiment Result of Three Systems on Different Message Rate

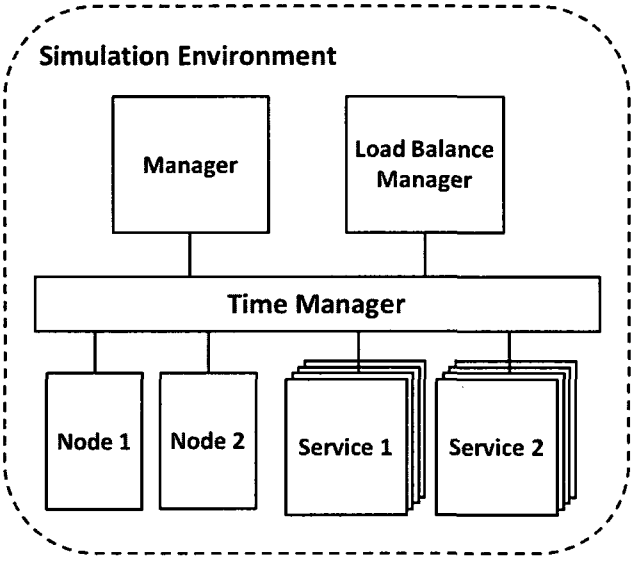


Figure 6.9: Simulation Environment

Manager is the main process, while the other entities are implemented as threads. Node 1 and Node 2 entities are represented as the computers and Service platform. Each Node entity has 8 pipelines, and the services have to occupy the pipelines to process the tasks. We tested our service migration component proposed above. It takes about 400ms in average for migrating an Object which is about 10KBs. We set the delay for migrating one service to 400ms. The tasks sending rate, number of services and QoS of services are exactly the same as in experiment 1. The simulation result is shown in Figure 6.10. It's very close to the result of our emulation experiment.

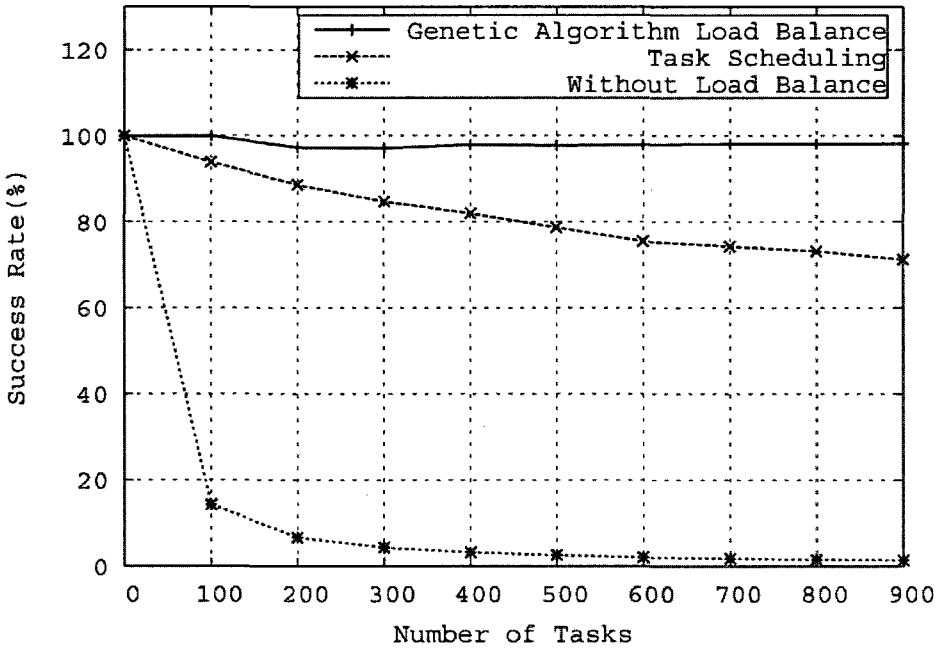


Figure 6.10: Simulation Result

6.3 Conclusion

We conclude that when the resource are limited our Tree-based QoS-aware Framework has better performance than the Flat-based framework in terms of task-handling capability and system resource utilization. On the other hand, a QoS-aware system without

any load balance solution is impractical. A system with a Task Scheduling Algorithm can perform much better. However, to maximize the usage of system resources, the Services Migration Algorithm should be introduced into the system. However, currently, most load balancing approaches on QoS pay more attention to message routing and traffic engineering. We propose a load balance algorithm with a Task Scheduling Algorithm on dynamic QoS properties and a Services Migration Algorithm on Genetic Algorithm.

Chapter 7

Conclusion and Future Work

In this chapter, we will summarize the main points of this thesis and briefly introduce future work on this research topic. The main components of this thesis consist of three parts: a JXTA Message Layer infrastructure for the designing and testing of SOA systems, a Tree-based QoS-aware service composition framework, and a load balancing scheme designed for this tree-based QoS-aware framework.

In Chapter 2, we first discussed the several modeling and simulation techniques of SOA system. However, these tools are unable to work as real infrastructure for the SOA system. They are only simulation tools. Later we introduced the existing techniques of service composition and the relevant load balancing scheme on the QoS-aware system. Web service and general P2P services are both feasible approaches for implementing the QoS-aware system. Because of the disadvantages of web service, its poor performance when compared to other distributed computing techniques and security issue caused by the utilization of HTTP, we decided to implement our design as P2P services. We also needed to decide whether to design our system as a distributed system or a hierarchical system. Hierarchical system can have a global view of system easily, which may simplify the management of the system. Our purpose is to design a QoS-aware system, which can supply appropriate services to users. Thus, we need to monitor and manage the service components and extract enough information to supply appropriate services to

users. Consequently, our choice of system structure is the hierarchical system.

Chapter 5 presents a JXTA Message Layer infrastructure for achieving SOA modeling with the JXTA P2P framework. We designed and implemented a message layer infrastructure with JXTA for message exchanging and building up network connections. We improved some features of JXTA to enhance the performance and simplify the usage of JXTA API. This framework is not only for SOA modeling. It also can be used on the infrastructure of real P2P system.

In Chapter 3, we proposed a design for Tree-based QoS-aware service composition and management framework; this is built on JXTA Message Layer infrastructure, as demonstrated in Chapter 5. We compared our design to a commonly used flat-based service composition and management system, in terms of user requests' level of satisfaction. The result shows that our design satisfies user requests more than the commonly used flat-based system.

In order to implement scalability on our Tree-based system in Chapter 3, we designed a load balancing scheme in Chapter 4. This load balancing scheme consists of two parts, a task scheduling algorithm that uses dynamic QoS properties and a Genetic Algorithm based service migration scheme. We compared our load balancing scheme to a scheme without any load balancing techniques and to another scheme with only task scheduling. The result shows that a QoS-aware system without any load balancing techniques is impractical. A system with a task scheduling algorithm can work much better to maximize the usage of system resources. When researchers work on the approaches for service composition, they should also consider task scheduling. Most research on service composition tries to discover services with the best composition. However, this is impractical due to the limited resources. We should provide appropriately composed services for user request to maximize the satisfaction of users. Service migration should be introduced into systems for scalability. Task scheduling techniques only solve problems for imbalanced workloads among services. However, in order to solve the problem on imbalanced workload on nodes, we need service migration techniques to exchange services between

heavily loaded nodes and lightly loaded nodes.

One topic of interest for future research is the integration of the QoS-aware system with simulation tools for real-time simulation, like [6]. [5] shows a multi-layered JXTA-core Service Oriented Architecture, which can support dynamic configuration and real-time capable distributed simulation infrastructure. This architecture consists of a User Access Layer that supports different types of user application for accessing the system, a User Application Layer that provides a set of services and proxy services for the exchange of information between the other two layers, and a core distributed simulation engine layer for controlling the simulation. Our design of the QoS-aware system is able to integrate with the User Application Layer in this simulation infrastructure to provide specific services and to guarantee performance.

Another topic of interest is to apply our research to emergency response systems, which require fast and reliable reactions upon request. Our idea is to design a decision maker based on the distributed simulation. This decision maker is a semi-automatic or automatic system. It is able to predict future conditions based on the input data of the current situation. Based on these predictions, it can make proper decisions to respond to the situation. One of the challenges of this research is to guarantee a figure for the amount of time used for the collection of current data and the response time. This is the reason why we introduced QoS-aware system techniques to the emergency response system.

Bibliography

[1] I. Ahmad and J. Kamruzzaman. Preemption-aware instantaneous request call routing for networks with book-ahead reservation. *IEEE Transactions on Multimedia*, 9(7):1456 – 1465, 2007.

[2] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369 – 384, 6 2007.

[3] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):65 – 104, 2 1999.

[4] E. Borcocil, A. Asgari, N. Butler, T. Ahmed, A. Mehaoua, G. Kourmentzas, and S. Eccles. Service management for end-to-end QoS multimedia content delivery in heterogeneous environment. In *Proceedings of Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/ E-Learning on Telecommunications Workshop*, pages 46–52, 2005.

[5] A. Boukerche and M. Zhang. Towards peer-to-peer based distributed simulations on a grid infrastructure. In *Proceedings of Simulation Symposium, ANSS 2008*, pages 212 – 219, 2008.

[6] A. Boukerche, M. Zhang, and H. Xie. An efficient time management scheme for large-scale distributed simulation based on JXTA peer-to-peer network. In *Proceeding*

of 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, pages 167–172, Vancouver, BC, Canada, 2008.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villan. A lightweight approach for QoS-aware service composition. In *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, 2004.

[8] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069 – 1075, 2005.

[9] D. A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co., Singapore, 1999.

[10] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE transaction on Internet Computing*, 6(2):86–93, 3 2002.

[11] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29 – 34, 10 2003.

[12] L. Du, J. Bigham, L. Cuthbert, P. Nahi, and C. Parini. Intelligent cellular network load balancing using a cooperative negotiation approach. In *Proceedings of Wireless Communications and Networking*, pages 1675 – 1679, 2003.

[13] L. Du, J. Bigham, L. Cuthbert, C. Parini, and P. Nahi. Cell size and shape adjustment depending on call traffic distribution. In *Proceedings of Wireless Communications and Networking conference*, pages 886 – 891, 2002.

[14] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Indianapolis, IN, USA, 2005.

[15] K. Gilly, S. Alcaraz, C. Juiz, and R. Puigjaner. Service differentiation and QoS in a scalable content-aware load balancing algorithm. In *Proceedings of 40th Annual Simulation Symposium*, pages 185 – 193, 2007.

[16] A. Goderis, P. Li, and C. Goble. Workflow discovery: the problem, a case study from e-science and a graph-based solution. In *Proceedings of IEEE International Conference on Web Services*, pages 312 – 319, 2006.

[17] W3C Working Group. Web services architecture, August 2008. `http://www.w3.org/`.

[18] X. Gu and K. Nahrstedt. *QoS-Aware Service Composition for Large-Scale Peer-to-Peer Systems*, chapter 24, pages 395 – 410. Kluwer Academic Press, 2003.

[19] X. Gu and K. Nahrstedt. Distributed multimedia service composition with statistical QoS assurances. *IEEE Transactions On Multimedia*, 8(1):141 – 151, 2 2006.

[20] X. Gu, K. Nahrstedt, and B. Yu. SpiderNet: an integrated peer-to-peer service composition framework. In *Proceedings of 13th IEEE International Symposium on High performance Distributed Computing*, pages 110 – 119, 2004.

[21] R. Van Haalen, R. Malhotra, and A. De Heer. Optimized routing for providing ethernet LAN services. *IEEE Communications Magazine*, 43(11):158 – 164, 11 2005.

[22] F. Hayes-Roth. Review of adaptation in natural and artificial systems by john h. holland. *ACM SIGART Bulletin*, (53):15 – 15, 8 1975.

[23] Sun Microsystems Inc. JXSE 2.5 programmers guide: JXTA concepts, July 2009. `https://jxta-guide.dev.java.net/`.

[24] Sun Microsystems Inc. JXTA v2.0 protocols specification, July 2009. `https://jxta-spec.dev.java.net/`.

[25] P. Jiang, J. Bigham, and J. Wu. Providing enhanced QoS differentiation to customers using geographic load balancing. In *Proceedings of The 9th European Conference on Wireless Technology*, pages 154 – 157, 2006.

[26] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *Proceedings 12th International Conference of Advanced Information Systems Engineering*, pages 431–445, 2000.

[27] S. Kim and P. K. Varshney. Adaptive load balancing with preemption for multimedia cellular networks. *IEEE Wireless Communications and Networking*, 3:1680 – 1684, 3 2003.

[28] S. Kim and P. K. Varshney. An integrated adaptive bandwidth-management framework for QoS-sensitive multimedia cellular networks. *IEEE Transactions on Vehicular Technology*, 53(3):835 – 846, 5 2004.

[29] A. Kogekar, D. Kaul, A. Gokhale, P. Vandal, U. Praphamontripong, S. Gokhale, J. Zhang, Y. Lin, and J. Gray. Model-driven generative techniques for scalable performability analysis of distributed systems. In *Proceeding of 20th Parallel and Distributed Processing Symposium*, 2006.

[30] A. Korostelev, J. Lukkien, J. Nesvadba, and Y. Qian. QoS management in distributed service oriented systems. In *Proceedings of the 25th IASTED International Multi-Conference*, pages 345–352, 2007.

[31] L. S. Krause, L. A. Lehman, B. R. McQueary, A. P. Stirtzinger, and S. A. Stirtzinger. The role of intelligent models in the implementation of service oriented architectures. In *Proceeding of International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 63–68, 2007.

[32] C. M. Lagoa, H. Che, and B. A. Movsichoff. Adaptive control algorithms for decentralized optimal traffic engineering in the internet. *IEEE/ACM Transactions on Networking*, 12(3):415–428, 6 2004.

[33] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana: a graphical web service composition and execution toolkit. In *Proceedings of IEEE International Conference on Web Services*, pages 514 – 521, 2004.

[34] B.A. Movsichoff, C.M. Lagoa, and H. Che. Decentralized optimal traffic engineering in connectionless networks. *IEEE Journal on Selected Areas in Communications*, 23(2):293 – 303, 2 2005.

[35] B.A. Movsichoff, C.M. Lagoa, and H. Che. End-to-end optimal algorithms for integrated QoS, traffic engineering, and failure recovery. *IEEE/ACM Transactions on Networking*, 15(4):813 – 823, 8 2007.

[36] N.Milanovic and M. Malek. Current solutions for web service composition. *IEEE Transactions On Internet Computing*, 8(6):51–59, 2004.

[37] M. Oh, J. Baik, S. Kang, and H. Choi. An efficient approach for QoS-aware service selection based on a tree-based algorithm. In *Proceedings of The IEEE Conference on Computer and Information Science*, pages 605 – 610, 2008.

[38] E. Sloane, T. Way, V. Gehlot, R. Beck, J. Solderitch, and E. Dziembowski. A hybrid approach to modeling SOA systems of systems using cpn and mesa/extend. In *Proceeding of 1st Systems Conference*, pages 1–7, 2007.

[39] I. Taylor, M. Shields, and I. Wang. Distributed P2P computing within Triana: a galaxy visualization test case. In *Proceedings of Parallel and Distributed Processing Symposium*, page 8, 2003.

[40] I. Taylor, M. Shields, and I. Wang. *Grid Resource Management: State of the Art and Future Trends*, chapter 27. Kluwer Publishing, 2003.

[41] S. Vale and S. Hammoudi. Model driven development of context-aware service oriented architecture. In *Proceeding of 11th IEEE International Conference on Computational Science and Engineering Workshops*, pages 412 – 418, 2008.

[42] H. Wada, J. Suzuki, and K. Oba. Modeling non-functional aspects in service oriented architecture. In *Proceeding of IEEE International Conference on Services Computing*, pages 222–229, 2006.

[43] H. Wada, J. Suzuki, and K. Oba. A feature modeling support for non-functional constraints in service oriented architecture. In *Proceeding of IEEE International Conference on Services Computing*, pages 187–195, 2007.

[44] Z. Wu and J. Luo. QoS-resource graph model for web service composition in service oriented computing. In *Proceedings of Sixth International Conference on Grid and Cooperative Computing*, pages 411 – 416, 2007.

[45] H. Xie, A. Boukerche, M. Zhang, and B. P. Zeigler. Design of a QoS-aware service composition and management system in peer-to-peer network aided by devs. In *Proceeding of 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 285–291, Vancouver, BC, Canada, 2008.

[46] U. Yildiz and C. Godart. Information flow control with decentralized service compositions. In *Proceedings of IEEE International Conference on Web Services*, pages 9 – 17, 2007.

[47] B. P. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation.* Academic Press, New York, NY, USA, 2000.

[48] F. Zeng and Z. Chen. Load balancing placement of gateways in wireless mesh networks with QoS constraints. In *Proceedings of The 9th International Conference for Young Computer Scientists*, pages 445 – 450, 2008.

[49] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions On Software Engineering*, 30(5):311, 2004.