Supporting Distributed Transaction Processing Over Mobile and Heterogeneous Platforms

A Dissertation
Presented to
The Academic Faculty

By

Wanxia Xie

In Partial Fulfillment
Of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Georgia Institute of Technology
December 2005

Supporting Distributed Transaction Processing Over Mobile and Heterogeneous Platforms

Approved By:

Dr. Shamkant B. Navathe, Advisor
College of Computing
Georgia Institute of Technology

Dr. Leo Mark
College of Computing
Georgia Institute of Technology

Dr. Edward Omiecinski
College of Computing
Georgia Institute of Technology

Dr. Sushil K. Prasad
Department of Computer Science
Georgia State University

Dr. Wai Gen Yee
Department of Computer Science
Illinois Institute of Technology

Date Approved: November 7, 2005

To my father, Xiao-Zhang Xie,

for his guidance, support and sacrifice.

To my mother, brother and sisters,

for their support, patience, and encouragement.

To my wife, Wenzhuo Guo,

for her love, support and sacrifice.

ACKNOWLEDGEMENTS

My greatest gratitude goes to my advisor Dr. Shamkant B. Navathe for his guidance and consistent support. His knowledge, perceptiveness, and inspiring discussions have guided me throughout my Ph.D. study. I also want to express my sincere gratitude to my co-advisor Dr. Sushil K. Prasad. He provided encouragement, sound advice, good company, and lots of good ideas. It was such a pleasure to work with them for the past few years; they always support me and lead me to the right direction. Without their guidance and encouragement, this thesis would not have been possible.

I would also like to thank other members of my committee, Dr. Leo Mark, Dr. Edward Omieciski, and Dr. Wai Gen Yee for their interests in my work. Their insightful comments have significantly affected the substance and presentation of my work.

I would also like to thank the faculty members from Georgia State University, Dr. Anu Bourgeois, Dr. Erdogan Dogdu, Dr. Yi Pan and Dr. Raj Sunderraman for their straightforward advices.

My fellow students made my life at Georgia Tech an enjoyable experience. I wish to thank, Rocky Dunlap, Fariborz Farahmand, Meng Guo, Weiyun Huang, Ying Liu, Minh Quoc Nguyen, Saurav Sahay, Hao Wu, Donghua Xu, Wenrui Zhao, for their friendship, encouragement, helpful discussions and all the good times we spent together. In addition, I want to thank students in GEDC DiMoS group, Akshaye Dhawan, Arthi Hariharan, Janaka Lalith and Srilaxmi Malladi, for their helpful discussions.

Finally, I am forever indebted to my parents, brother and sisters for their endless support, patience, and encouragement. I would trade anything if my father could attend my graduation ceremony. Words could not describe how much I appreciate his wisdom, guidance, support and sacrifice. Rest in Peace, Father. I owe special thanks to my wife Wenzhuo Guo for her love, sacrifice and support in these years. Without their support, this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Internet, pervasive computing, peer-to-peer computing, and related developments have opened up vast opportunities for developing collaborative applications. To benefit from these emerging technologies, there is a need for developing techniques that will allow deployment of these applications on heterogeneous platforms as well as to provide the tools necessary for the development of these applications. To meet this challenging task, we need to address the characteristics of mobile peer-to-peer systems such as frequent disconnections, frequent network partitions, difficult central control, peer heterogeneity, and group collaboration. This research is aimed at developing the necessary models, techniques and algorithms that will enable us to build and deploy such applications in the Internet-enabled mobile peer-to-peer environments.

Keeping the above in mind, this research sets out to investigate the following important underlying problems related to supporting distributed transactions over mobile peer-to-peer environments:

1. We propose a multi-state transaction model for transaction processing over a collection of heterogeneous, possibly mobile data stores. Based on this model, we develop a quality aware transaction processing framework to incorporate quality of service with transaction processing. In order to support quality aware transactions, we redefine and adapt the traditional concepts such as atomicity, consistency, isolation and durability to suit the proposed environment so that transactions can be made flexible and adaptable. We also develop a quality specification language to associate quality of service with transaction properties.

On the whole, we support disconnection-tolerant, and partition-tolerant transaction processing without assuming that a central server is always available.

2. Based on the above quality aware transaction processing framework, we develop a probabilistic concurrency control mechanism and group based transaction commit protocol in which we utilize the probability and feedback to adaptively find the balance between the cascading abort and the long blocking in concurrent transaction control. This reduces blockings in transactions and improves the transaction commit ratio.
3. Based on our transaction processing framework, we explore several transaction processing architectures based on our transaction model in order to support distributed transactions over dynamically partitioned networks.
4. We develop a scalable, efficient and highly available directory service called PeerDS to support the above framework as many mobile devices could provide services and participate in a distributed transaction. We address the scalability and dynamism of the directory service from two aspects: peer-to-peer and push-pull hybrid interfaces. We also address peer heterogeneity and load balancing in the peer-to-peer system. We optimize the routing algorithm in a virtualized P2P overlay network and develop a generalized Top-K server selection algorithm for load balancing, which could be optimized based on different factors such as proximity and cost. From push-pull hybrid interfaces aspect, we propose to add a push interface, in addition to the conventional pull interface, to reduce the overhead of directory servers caused by frequent queries of directory clients.

5. We develop and evaluate different filter indexing schemes to improve the scalability and update scheduling of large subscription-based systems (i.e., the push-pull hybrid interfaces in PeerDS) as the filtering process of the updates might become the bottleneck.

These techniques extend the capabilities of key components of our System on Mobile Devices (SyD) middleware, which enables collaborative distributed applications over heterogeneous mobile handheld device and data stores.

CHAPTER 1 INTRODUCTION

Internet, pervasive computing, peer-to-peer computing, and related developments have opened up vast opportunities for developing collaborative applications. To benefit from these emerging technologies, there is a need for developing techniques that will allow deployment of these applications on heterogeneous platforms as well as to provide the tools necessary for the development of these applications. This research is aimed at developing the necessary models, techniques and algorithms that will enable us to build and deploy such applications in the Internet enabled, mobile peer-to-peer environments of the future.

Integration of mobile devices with collaborative applications is crucial for the following reasons (apart from the conventional reasons related to availability of heterogeneous and collective computing and communication resources):

(i) The most current information resides on mobile devices: e.g., user specific data like the location, users' decisions such as schedules, policies and any other user parameters.

(ii) Synergy brought in by the multitudes of handheld devices needs to be leveraged by Internet/Grid infrastructures (e.g., disaster evacuation re-routing on a college campus based on collective perception and real-time collaboration).

(iii) A backup networking infrastructure can be provided by mobile server devices, which is crucial in case of sabotage or transient outage of the power grid or the backbone network.

For example, if handheld devices of staff and police are enabled to serve as ad-hoc network nodes in sports stadiums, college campuses, airports or malls, emergency re-routing in case of a backbone outage can still be performed and people can employ their mobile devices to query real-time evacuation information pertaining to their location in the format/language of their choice. Rapid incremental reprogramming to effect structural changes in the collaborative applications can help when response is needed to emergencies/disasters or emerging situations.

The tragic events of September 11 powerfully demonstrate the need for such a system. Despite the fact that New York City already had a disaster plan, their ability to handle the attack was limited. Their disaster coordination base, located under the Twin Towers was destroyed, as was the Verizon telephone switching station nearby. The loss of these two facilities crippled the ability of the authorities to effectively coordinate personnel and resources, with possibly tragic consequences.

The subsequent blackout in New York City in the summer of 2003 demonstrated both failure and success in emergency response. Again, cellular telephone service was mostly unavailable [9], however, the relative grace with which the city recovered was a testament to the city's improved emergency response, among other things. However, a mayoral task force indicated that the city still has far to go to be able to handle more severe events [48].

Emergency management and response (or lack of it) in New Orleans City in the recent Hurricane Katrina further demonstrates the necessity of such systems again. When Hurricane Katrina hit New Orleans, all backbone systems for communication crashed. When the levees breached, the local officials could not pass this message out of the city. The evacuation plan was useless in the first week as the control center could not collect the latest information from the officials in the front areas and make the best decision based on this information. Moreover, they could not send the decisions to the officials immediately. The difficulty in the collaboration caused the loss of thousands of lives and billions of dollars.

These factors make for a strong case for putting server functionality on mobile devices – this can usher in the quantum change needed for the next generation of collaborative applications. To integrate a variety of heterogeneous, possibly mobile, devices with varying data formats, operating systems, network protocols, and computing and communications capabilities, an *adaptive yet uniform view* is a must. Typical heterogeneous environments would include legacy applications, and mobile devices with limited resources and weak connectivity. We need to develop techniques to enable handheld servers to support multiple transaction states and maintain consistency despite disconnection.

1.1 Mobile Peer-to-Peer Systems

With mobile devices enabling server capabilities, mobile peer-to-peer systems are becoming the main platform for collaborative applications.

There are two kinds of mobile networks: *infrastructure based mobile network* and *infrastructure-less mobile network* [19]. Figure 1 describes infrastructure based mobile network. Mobile units, i.e., mobile devices, are connected to the high-speed wired network through Base Stations. If a mobile unit M_1 wants to communicate to another mobile unit M_2, it will send the request to its current base station B. If B also manages M_2, M_1 can communicate with M_2 through B now. Otherwise B will contact the corresponding base station B' that manages M_2, the communication path will be M_1<->B<->B'<-> M_2, in which B communicate with B' through the backbone network.

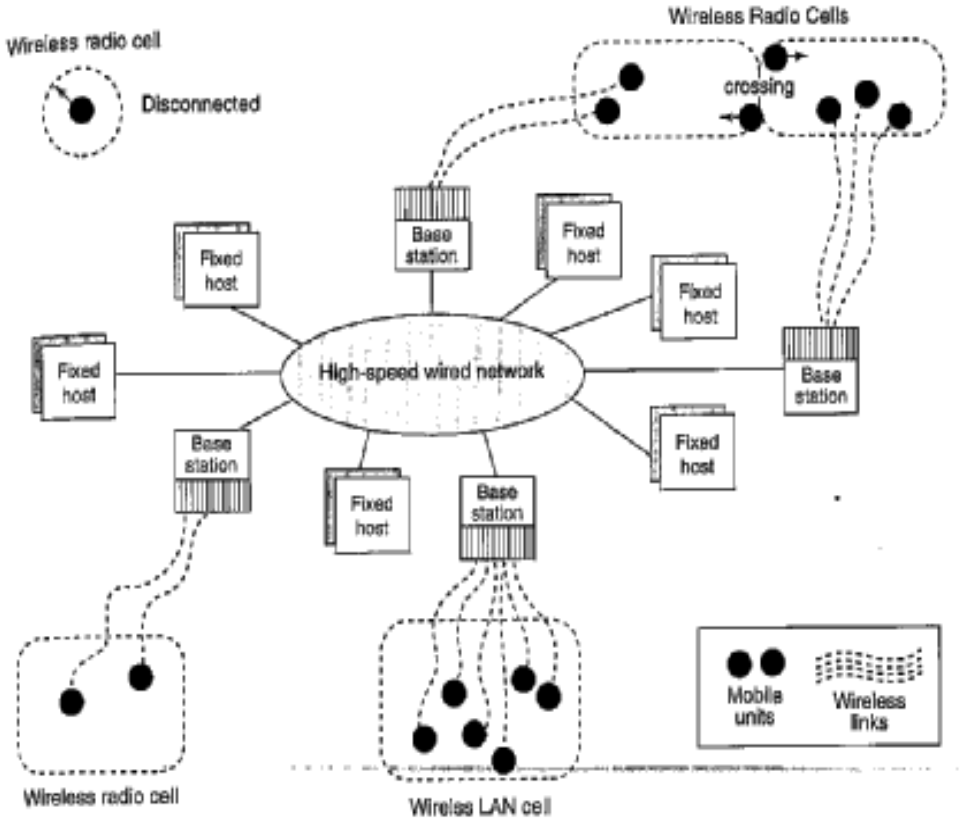


Figure 1. Infrastructure Based Mobile Network

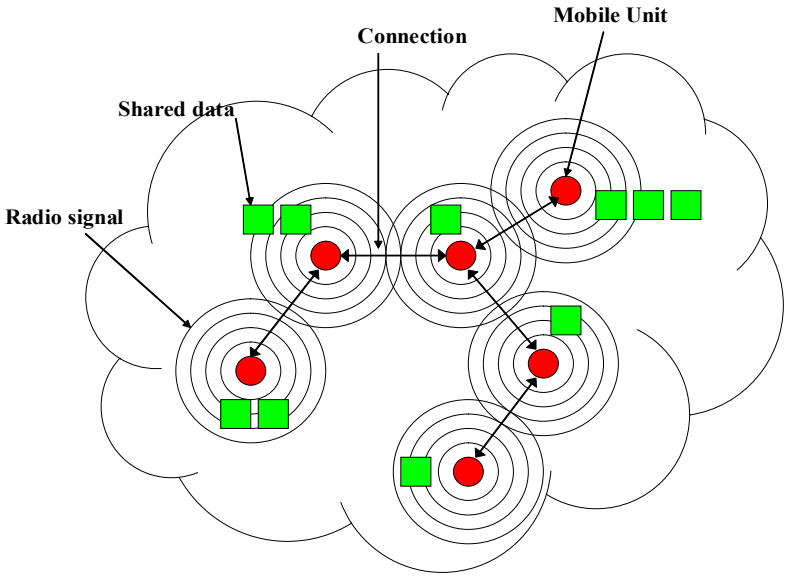


Figure 2. Infrastructure-less Mobile Network

Figure 2 describes infrastructure-less mobile network. Co-located mobile units do not need to communicate via a backbone network, instead they communicate with each other though cost-effective technologies such as Bluetooth. When two mobile units are not connected directly, the communication messages/data will be routed through other mobile units.

There are also hybrid mobile networks in which an infrastructure based mobile network is extended at the cell with an infrastructure-less based mobile network. A Bluetooth-enabled cell phone could communicate with the base station through infrastructure based mobile network, but it could also communicate with other mobile devices such as PDA through Bluetooth. Mobile peer-to-peer systems addressed in this

research could be based on either architecture. To differentiate from previous researches, we focus on *infrastructure-less mobile network* and *hybrid mobile network*.

Peer-to-peer architecture is a type of network in which each peer has the same responsibilities. This is contrasted to client/server architecture where servers are dedicated to serving clients. There are two kinds of peer-to-peer networks: *hybrid peer-to-peer network* and *fully peer-to-peer network*. In hybrid peer-to-peer network, there is a central directory server that could provide the information of all peers. Usually peers publish resource information in the directory server and query the directory server about resource information. Napster is a typical example for hybrid peer-to-peer network. In fully peer-to-peer network, there are no central servers. In the first generation fully peer-to-peer network such as Gnutella [29], querying and routing are done through broadcast, which cause message flooding in the network. In the second-generation fully peer-to-peer network, i.e., structured peer-to-peer networks such as Chord, CAN, Pastry and Tapestry [69][64][65][86], querying and routing are based on Distributed Hash Table. Structured peer-to-peer networks are guaranteed to reach each peer in a certain number of hops. For example, the upper bound of hops is O(logN) in which N is the total number of peers.

Mobile peer-to-peer systems addressed in this research could be based on either type of peer-to-peer networks. But we mainly focus on fully peer-to-peer network.

1.2 Characteristics of Mobile Peer-to-Peer Systems

Mobile networks and peer-to-peer computing discussed so far lead to the following main characteristics of mobile peer-to-peer systems:

- *Frequent disconnections*. We cannot assume that all mobile devices are always connected in the development of collaborative applications. We need to address the frequent, sometimes intentional, disconnections of participating devices.
- *Frequent network partitions*. Frequent disconnections and moving of mobile devices may cause frequent network partitions. Network partitions will be a very serious problem for collaborative applications.
- *Central control is difficult*. A central server for mobile peer-to-peer systems is not practical or possible. For example, a central server in case of network partitions will crash the devices in all but one partition.
- *Peer heterogeneity*. Naturally, there is heterogeneity among peers when we consider hardware and software including operating systems, language and databases available on each platform. In addition, peers are heterogeneous in computing capabilities such as CPU and storage, network bandwidth, and battery power. It would be essential to do load balancing based on their capabilities.
- *Collaboration based on group*. The applications targeted in mobile peer-to-peer systems could use collaboration and support decisions based on groups that may be formed in an ad-hoc manner.

1.3 **Motivating Applications**

In this section, we introduce a few categories of motivating applications: emergency applications in natural or non-natural disasters, battlefield applications, enterprise applications and personal applications. In addition, this research could also be applied to collaborative sensor data management.

- **Disaster Applications**

Disaster applications include emergency applications in earthquakes, blackout and especially terrorist attacks in the current political climate, where fixed and stable infrastructure is limited. In these disaster events, we cannot execute distributed transactions and we have no alternatives. One sample application is *Traffic Control and Emergency Application***.** Evacuation is usually complicated by unexpected obstacles (e.g., levee breaching, accidents, roadblocks, etc.). The information gathered from a variety of input sources and sensors (personal handhelds, servers, and sensors such as traffic and visual sensors), resulting in real-time distributed decision-making with no or minimal human intervention and disseminated through handhelds can alleviate traffic jams and pandemonium that are typical during unexpected disaster events. The key features of this application are: dynamic distributed gathering of information or events (fed automatically into the system); dynamic distributed triggering resulting in real-time update of evacuation and other emergency directions (centralized as well as distributed decision making backing up each other); location and device based emergency information.

- **Battlefield Applications**

More than often, the central control is not available in the front of the battle for various reasons such as security, geographical, malfunctioned devices and the destruction of the control center. No matter what happens, the soldiers need to collaborate with each other to win the battle. The commander-in-chief would like to learn the latest progress in all positions. How to support such applications in the battle environment is the problem that this research would like to address.

- **Enterprise Applications**

Typical enterprise applications of interest include surface transportation systems and airport systems. For surface transportation systems, there are many fleets that operate within a company and/or collaborate with fleets from other companies. Usually depots send instructions to trucks and trucks may send the latest information to depots and other neighboring trucks. Depots may also communicate with other depots for information changes and transactions. Depots and trucks form a mobile ad-hoc network. The typical activities among depots and trucks include scheduling and canceling, pickup and shipping and route scheduling. These activities are run as distributed transactions over mobile ad-hoc networks. A similar system exists in airports. Airlines want to communicate with customers and other airlines for scheduling. Customers may use their mobile devices to communicate with other customers for ticket trading (not technologically supported now) and get the latest information from airlines. All these activities involve distributed transactions over mobile ad-hoc networks and need to solve the same problems: How should we enable server functionality in mobile devices? How should we deal with disconnections and network partitions associated with these distributed transactions since the transactions would otherwise be aborted in traditional transaction processing techniques?

- **Personal Applications**

For personal applications, we mainly use activity scheduling application as examples. Let's look at a student's life on campus. Examples of such activities are fetching lunch from cafeteria, boarding campus shuttles, visiting health center, scheduling a meeting for

the advisory committee, etc. For example, a student can query a bus's mobile sever and know how long the waiting time is. In turn, the shuttles can use the location-based queries (source and destination information) and their aggregate numbers to reroute for faster service, possibly employing p2p collaboration to dispatch the shuttles to appropriate shuttle stands (note the implication of this scenario for dispatching police in the traffic/emergency application above). The same questions that we raised in enterprise applications apply in personal applications.

1.4 Research Theme and Problem Areas

In order to support the collaborative applications discussed above, we need to address the following problem areas.

1.4.1 Middleware

The current technology for the development of such collaborative applications over a set of wired or wireless devices and networks has a few limitations. It requires explicit and tedious programming on each kind of device, both for data access and for inter-device and inter-application communication. The application code is specific to the type of device, data format, and the network. The data store provides only a fixed set of services disallowing dynamic reconfiguration. Applications running across mobile devices become complex due to lack of persistence and weak connectivity. A few existing middlewares have addressed the stated requirements in a piecemeal fashion. Limitations include only client-side programming on mobile devices, a restricted domain of applications, or limited in group or transaction functionalities or mobility support.

Our ongoing project called Systems on Mobile Devices (SyD), described in Chapter 3, has been addressing the middleware issues mentioned above. A preliminary version of SyD has been prototyped on PDAs and cell phone emulators, and several samples of collaborative applications [55][56][57][58][59][60] have been developed. While SyD would provide us with a basic platform to serve as a test-bed and to incorporate a host of new ideas, the research will be equally applicable to all competing middleware and transaction processing infrastructures.

The future work remaining in the SyD middleware focus on integrating web services, supporting broader mobile devices and data stores, supporting Quality of Service at the different levels of applications.

1.4.2 Distributed Transactions

A transaction addressed in this research includes, but is not limited to, traditional transaction in the database system. It could be a task needing to have the full or partial transactional properties such as ACID (Atomicity, Consistency, Isolation and Durability).

- **Transaction Models for Mobile Heterogeneous Environments**

The existing transaction models [13][61][11][73][74][52][3] do not sufficiently capture the complexity and the execution requirements of mobile and/or Internet applications since most of the existing models are developed with a database-centric view [18]. Most of existing transaction models in mobile data management assume an infrastructure based mobile network and are not applicable to infrastructure less mobile networks. Besides, it is also assumed that transaction components are to be executed over applications that are cooperatively developed. New transaction models are needed to

address application requirement including a workflow specification that captures the collaborative yet independent, mobile and heterogeneous nature of application components.

Traditional transaction processing techniques also assume that broad network bandwidth, rich computing and storage resources are always available. With the characteristics of mobile peer-to-peer systems, traditional transaction processing techniques will be very challenging and need to address the following specific issues.

(i) A mobile transaction is expected to be long-lived, yet intermittent due to disconnection and energy issues in mobile devices. The quality of transaction service is sensitive and highly noticeable in this new environment especially in multimedia-related applications.

(ii) Transactions have different priorities because of their real-time requirements and/or ACID (Atomicity, Consistency, Isolation and Durability) properties.

(iii) A central coordinator might not be available for transactions.

(iv) A transaction might face frequent network partitioning.

As can be seen, the problems outlined above collectively call for a new transaction model for developing and executing collaborative applications in mobile peer-to-peer systems. The multi-state transaction model proposed in Chapter 4 will address this issue.

- **Quality of Service and Adaptability Issues with Transaction Models**

In heterogeneous and mobile systems, the availability of resources dynamically changes. The uncertainty of resources needs to be handled gracefully so that the quality of transactions can be monitored and improved. Currently almost all related Quality of

Service (QoS) research focuses on multimedia applications and little attention is paid to distributed transactions over heterogeneous and mobile platforms. Although prior QoS research provides precious knowledge and experience, it is not suitable for QoS in transactions since it does not have a proper transaction model and does not address the basic issue for transactions: adaptive ACID properties instead of adaptive transmission rates. In order to support quality aware transaction processing, we need to design a QoS specification language to associate adaptive ACID properties with different transaction states and different quality levels of transactions. We also need to implement code generators to automatically generate code based on QoS specification of a transaction. Furthermore, how should we measure the quality of a transaction? How could we adapt the quality of a distributed transaction while maintaining the correctness criteria? The quality aware transaction processing framework described in Chapter 4 will address these problems.

- **Concurrency Control and Transaction Commit Protocol**

As we learned in concurrency control, transactions share resources such as data items. To maintain the consistency and correctness of current transactions, techniques such as the two-phase locking protocols are used. The characteristics of mobile peer-to-peer systems such as frequent disconnection and frequent partitions indicate that the transaction abort ratio would be very high. Even if the multi-state transaction model is used, the blocking in the two phase locking would cause low system throughout and low transaction commit ratio. On the other hand, without enforcing two-phase locking mechanisms in the concurrency control, **"***Cascading Abort*"[6] would occur and waste

the previous transaction commit effort. So we need to find the balance between the long blocking and cascading abort.

In addition, disconnections and network partitioning in mobile peer-to-peer systems make the traditional commit protocols such as the two-phase commit protocol and the three-phase commit protocol ineffective. We need to design a disconnection-tolerant and partition-tolerant commit protocol for mobile peer-to-peer systems.

The probability based concurrency control mechanism and the group based transaction commit protocol proposed in Chapter 5 will address these problems.

In this area, the main contribution of this dissertation is described as follows. We propose a multi-state transaction model for transaction processing over a collection of heterogeneous, possibly mobile data stores. Based on this model, we develop a quality aware transaction processing framework to incorporate quality of service with transaction processing. We redefine and adapt the traditional concepts such as atomicity, consistency, isolation and durability so that transactions can be made flexible and adaptable. We also develop a quality specification language to associate quality of service with transaction properties. On the whole, we support disconnection-tolerant, and partition-tolerant transaction processing.

The future work remaining is to support rapid development of quality aware transaction services, provide the QoS specification template for different domains, and evaluate the techniques in real emergency response applications.

1.4.3 Distributed Transactions over Dynamically Partitioned Networks

A dynamically partitioned network is a network in which its partitions are dynamically changing. The target environment is a distributed system with frequent disconnection and frequent partitions among component sites. Individual membership of each partition dynamically changes. A participating node may leave one partition and join another. These may form "ad-hoc partitions" in which some partitions may be expected and even intentional and other partitions are totally unexpected. For example, in a disaster relief effort, nodes equipped with short-range radios may move out of range of one another. In a battlefield, equipment intended to provide connectivity might be compromised so that it becomes inoperable. In either scenario, the wide physical range of the deployed area may prevent end-to-end connectivity. Under these environments, most existing transaction processing techniques will fail to continue or complete the transactions since no consensus will be reached among participants in the partitioned network.

Previous research focuses on how we run the transactions in various partitions and how to reconcile the inconsistencies after the partitions reconnect. In the environment described above, we do not have a stable partition. How would we continue and complete transactions given this dynamic environment? Opportunistic and pessimistic approaches do not work since we may never be able to reach consensus among members in different partitions. That is, there may always be partitions in the systems and there may not be a global connected view in the system. In Chapter 8, we explore these problems in the

extension of quality aware transaction processing framework. We have defined some open problems for future work in this area.

1.4.4 Resource Discovery and Dissemination

As we enable mobile devices to provide services, we are meeting new challenges: How to manage these huge amounts of resources? How to efficiently find the desired resources (services in this case) from the vast resource libraries? How to support real time resource discovery and resource information update? Co-ordination among applications calls for a directory service that provides a variety of functions. So we turn to the study of directory service.

There are two main issues that we want to address in the study.

- **Scalability**: The directory service must be scalable with available resources and services. There are two problems associated with a central directory service: (i) A central directory server is the performance bottleneck of a large-scale distributed system; (ii) A central directory server is the single point of failure of the system. A network failure or a network partition involving the directory server usually affects a large number of clients. There are several proposals to try to solve these problems. A hybrid directory service such as IBM DCE distributed directory service addressed the problem (i) above, but problem (ii) was left out. Multiple replicated directory servers may solve problem (i) and relieve problem (ii). However, it needs large investments to set up multiple high capability replication servers. Previous fully distributed peer-to-peer systems such as Gnutella [29] may

solve both problem (i) and (ii). But they suffer from poor search performance including response time and quality of results.

- **Dynamism**: In a dynamic distributed environment, service information may require frequent updates to be kept up-to-date. For example, when a device with location-based applications moves to a new location, it may need to contact new servers (with which the device did not communicate before) to get the location-based services. In traditional directory services, the device itself has to be aware of the location changes and periodically query the directory server to get up-to-date results back.

PeerDS proposed in Chapter 6 addresses both above issues. In addition, we explore the problems of peer heterogeneity and load balancing, which is brought by PeerDS. Peer virtualization is utilized to address peer heterogeneity. But it causes serious overhead on routing and searching. In Chapter 6, we improve the routing protocol in virtualized peer-to-peer networks and propose a general optimized top-k peer selection algorithm for load balancing.

The future work remaining is to integrate with SyD middleware, integrate with quality aware transaction processing framework, deal with replication in P2P network, deal with fuzzy matching of queries in DHT based P2P network, and deal with node failures.

1.4.5 Scalability Issues in Large Subscription-Based Systems

In order to address the dynamism, PeerDS proposes a push interface in addition to the regular pull interface. As we study the above push interface in the directory service, one striking problem is the scalability issue of large subscription-based systems. Filtering is a popular approach to reduce information traffic in these subscription-based systems. Usually a subscriber sends its customized filter to the subscription management system and the system will filter the information being sent to the subscriber. Different subscribers have different filtering requirements. The same subscriber may have different filtering requirements at different times under different conditions. Filtering is especially important when a subscriber is located in a mobile device as it reduces the useless information received by the subscriber and hence reduces the energy consumption of the mobile device. On the other hand, it is difficult to develop a scalable subscription-based system, as the filtering process itself may become the bottleneck.

With the increasing number of subscribers and filters in the system, the filtering process becomes unmanageable and may constitute a bottleneck of the system. The system ends up repeating the same process for similar filters. The goal is to improve filter management and thereby to enhance the overall performance of the publisher/subscriber system by reducing the duplicate effort in the filtering process for similar filters in the system.

As there are frequent updates to the resource objects in a system, we need to schedule the updates being sent to subscribers. Without a proper filter management system, it takes a long time to find the real interests (the number of subscribers who are interested in the

update) of each update and this makes scheduling so difficult that most systems just send the updates for resource objects in a random order.

Filter indexing proposed in Chapter 7 will address these problems and give the overview of the solution. Our main contributions in this area are to develop and evaluate different filter indexing schemes to improve the scalability and update scheduling of large subscription-based systems. The future work remaining is to study complex filters such as statistics filters and investigate different update scheduling schemes.

1.5 Research Contributions

In this research, we have explored several issues concerning transaction processing over mobile peer-to-peer systems. This study is not intended to be exhaustive in that we do not claim to address all outstanding issues and problems in mobile peer-to-peer systems. However we rather hope that our study serves to provide insights into certain aspects of mobile peer-to-peer systems as well as to motivate additional work in this area. Through this research, we make several contributions:

1. We propose a multi-state transaction model for transaction processing over a collection of heterogeneous, possibly mobile data stores. Based on this model, we develop a quality aware transaction processing framework to incorporate quality of service with transaction processing. In order to support quality aware transactions, we redefine and adapt the traditional concepts such as atomicity, consistency, isolation and durability to suit the proposed environment so that transactions can be made flexible and adaptable. We also develop a quality specification language to associate quality of service with transaction properties.

On the whole, we support disconnection-tolerant, and partition-tolerant transaction processing without assuming that a central server is always available.

2. We develop a probabilistic concurrency control mechanism and group based transaction commit protocol in which we utilize the probability and feedback to adaptively find the balancing point between cascading abort and the long blocking in concurrent transaction control. This reduces blockings in transactions and improves the transaction commit ratio.
3. We explore several transaction processing architectures based on our transaction model in order to support distributed transactions over dynamically partitioned networks.
4. We develop a scalable, efficient and highly available directory service called PeerDS to support the above framework as many mobile devices could provide services and participate in a distributed transaction. We address the scalability and dynamism of the directory service from two aspects: peer-to-peer and push-pull hybrid interfaces. We also address peer heterogeneity and load balancing in the peer-to-peer system. We optimize the routing algorithm in a virtualized peer-to-peer overlay network and develop a generalized Top-K server selection algorithm for load balancing, which could be optimized on different factors such as proximity and cost. From push-pull hybrid interfaces aspect, we propose to add a push interface, in addition to the conventional pull interface, to reduce the overhead of directory servers caused by frequent queries of directory clients.

5. We develop and evaluate different filter indexing schemes to improve the scalability and update scheduling of large subscription-based systems (i.e., the push-pull hybrid interfaces in PeerDS) as the filtering process of the updates might become the bottleneck.

These techniques extend the capabilities of key components of our System on Mobile Devices (SyD) middleware, which enables collaborative distributed applications over heterogeneous mobile handheld device and data stores.

1.6 Roadmap for the Dissertation

The rest of the dissertation is organized as follows. Section 2 introduces current state-of-the-art. Section 3 gives the overview of the middleware SyD (System on Mobile Devices). Section 4 describes the quality aware transaction processing framework. Section 5 proposes a probabilistic concurrency control mechanism and the group based transaction commit protocol. Section 6 describes the investigation for a scalable peer-to-peer directory service and the solution for addressing peer heterogeneity and load balancing. In Section 7, we present the study of four filter indexing schemes for large subscription-based systems. In Section 8, we introduce some potential future works. Section 9 concludes the dissertation.

CHAPTER 2 RELATED WORK

This chapter provides an overview of related work in the areas of middleware, transaction models, concurrency control and transaction management, quality of service, distributed hash table based peer-to-peer systems, load balancing techniques, and indexing techniques in large subscription based systems.

2.1 Middleware

In this section we review several related middleware systems for mobile intelligent devices. Generally, these systems can be classified into P2P-protocol oriented systems and dynamic distributed applications (e.g. JXTA) or IP-based client-server applications (Jini, Microsoft .NET, IBM WebSphere Everyplace Suite). A large body of work in the heterogeneous database integration area has largely remained in the research domain where no specific products can be named.

JXTA [75] is a set of open, generalized P2P protocols that allows any connected device on the network from cell phone to PDA, from PC to server to communicate and to collaborate as peers. Currently, JXTA provides a way of peer-to-peer communication at the level of socket programming.

Proem [36][37] is another mobile peer-to-peer platform that supports developing and deploying mobile peer-to-peer applications. Compared to JXTA, Proem is geared more toward supporting mobile applications characterized by immediate physically proximal peers. System on Mobile Devices (SyD), described in Chapter 6, goes further in this

arena by focusing on general collaborative applications involving intensive database operations and complex business logic. In contrast to Proem, SyD relies on proxies and provides mechanisms for reusing existing SyD applications and services.

Jini [49], a more mature system compared to JXTA uses Java's Remote Method Invocation (RMI). Jini's limitations include the lack of scalability: Jini was originally designed for resource sharing within groups of about 10 to 100, and that at least one machine on the network is required to run a full Java technology-enabled environment.

Qualcomm's Binary Runtime Environment for Wireless (BREW) allows development of a wide variety of handset applications that users can download over carrier networks onto any enabled phone. Microsoft's .Net is a platform based on Web Services built using open, Internet-based standards such as XML, SOAP and HTTP. Communication among .NET Web Services is achieved through SOAP message passing. IBM WebSphere provides the core software needed to deploy, integrate and manage e-business applications. Web Sphere Everyplace Suite extends them to PDA's and Internet appliances. It supports client-side programming through WAP and allows the creation of discrete groups of users.

SyD supersedes the above technologies in terms of unique features such as its orientation toward mobile-specific applications, easy application to mobile devices, heterogeneity of data, simple middleware API, heterogeneous software/hardware, etc. Only SyD supports a normal database transaction model [60].

2.2 Transaction Models

The clustering model [52][53] developed at Purdue University provides strong and weak transaction states to deal with transactions in disconnection. The strong transaction state meets ACID guarantees as in traditional transactions. The weak transaction state on the other hand relaxes ACID guarantees and usually does read-only operations or local read/write operations. The weak transaction state is a great advance since it improves the availability and performance of transactions.

Kangaroo [15] provides a transaction model to capture both data and movement behavior. They do not consider any adaptation in the model. The open nested transaction model [8] developed at University of Pittsburgh utilizes sub-transactions to improve the performance of transactions and develops a compensation action for each sub-transaction to maintain atomicity. They did not provide support for collaboration among mobile devices.

Semantic-based transaction processing model [73] is proposed to facilitate autonomous disconnected operations in mobile database applications. A new class of objects called fragmentable objects has been defined. Fragmentable object can be split among a number of sites, operated upon independently and recombined in a semantically correct fashion (stacks, sets, queues, etc.) Most commonly used semantic property of operations is commutativity.

The multi-database model [84] investigates how multi-database transactions can be submitted from mobile devices. In this architecture a transaction coordinator stands on the fixed network and transactions are submitted to the transaction coordinator. The

transaction coordinator acts on behalf of the mobile device once the transaction is submitted. This enables to tolerate disconnections. Each database server in this architecture has a layer called Multi-database engine defined on top of the conventional transaction processing system. MDS engine takes care of all the coordination required to execute the transaction correctly. MDS has a message and transaction queue which enables asynchronous communication between mobile units. A new asynchronous RPC protocol has been proposed.

The toggle Transaction model [12] is based on multilevel transaction model with a set of transactions that can be compensated. [44] proposes isolation only transactions for mobile environment. Most recently, a timeout based transaction commit protocol is proposed by [35]. Mobile transaction management in Mobisnap [61][62] proposes reservation based transactions. A mobile transaction processing model using separate processing for read only transactions is proposed in [13].

[25][21] proposes semantic consistency and semantic atomicity to relax the traditional ACID properties. [45] provides a hierarchical view of ACID properties. 0 proposes a model to implement relative serializability. [67] proposes transaction groups to support cooperation among groups. All these relaxation techniques use semantic mobility that typically occurs in mobile distributed transactions.

Long-duration transactions [28] describe transactions that need a long time to process. One assumption for this long-duration transaction is that the central coordinator knows the state information of processes at all participants. In mobile peer-to-peer systems, such

assumption cannot be made as frequent disconnections and network partitioning make this a very difficult task.

Sagas [26] tries to address the locking in long-duration transactions. A transaction is composed of many sagas. Each saga is a collection of actions. Each action in a saga has a compensating transaction, which could reverse the action. In practice, this is very difficult since a compensating transaction is needed for each action. In addition, an action cannot be compensated in many cases.

The two-phase commit protocol [40] is the mainstream protocol to handle transaction commit process. All participants might be blocked when the central commit coordinator and a participant fail. The three-phase commit protocol [68] addresses the blocking situation by inserting an extra phase called “precommit phase”. However both protocols assume a central coordinator and cannot handle frequent disconnections and network partitioning.

OPT [33] proposes an optimistic commit protocol that allows transactions to “optimistically” borrow uncommitted data from one level transaction to address data blocking. It aborts two transactions at the worst case.

Due to the strong assumptions or architectural requirements, most of such models have limited use in practice. All these transaction models mentioned above do not support multiple transaction states and transaction adaptation. They do not support QoS for transaction processing.

One problem with current transaction models proposed in mobile systems is that they assume an infrastructure base mobile network and require the mobile support station

(MSS) in their models. With infrastructure less mobile networks, this assumption is no longer valid.

Another problem is that these models assume that the operations in mobile devices belong to the client. With the advance in wireless technologies, mobile devices can easily serve as servers. Researches need to put some effort on designing models so that they will provide server functions to mobile devices. Currently we do not have any transaction support for mobile peer-to-peer systems. Our QoS-aware transaction processing techniques, described in Chapter 4 and 5, will fill in this area.

2.3 Quality of Service in Transaction Processing

Until now, most quality of service research is studied in multimedia systems and networking. To our best knowledge, this is no previous research that incorporates quality of service into transaction processing.

Quality Object [42][43][38][72] in BBN Technologies provides a framework for applying Quality of Service (QoS) in distributed object applications. Their focus is integrating QoS with distributed object architectures such as CORBA. Cactus [5][32] at the University of Arizona provides a framework to support customizable fine-grained QoS development. They use micro-protocols to compose into a new QoS protocol. Gaia [47][41] at University of Illinois provides QoS support for multimedia applications. They use a control model to control the adaptation process.

As we discussed in Section 1.4.3, previous QoS research is not suitable for QoS in transactions since it does not have a proper transaction model and does not address the basic issue for transactions: adaptive ACID properties instead of adaptive transmission

rates. Our focus is to integrate QoS with transactions in mobile and heterogeneous systems. Quality aware transaction process framework in Chapter 4 will describe the details.

2.4 Distributed Hash Table Based Peer to Peer Systems

Consistent Hash [34] proposes the consistent hash scheme in which the hashed value does not change with joining and leaving of peer nodes. Based on consistent hash, Chord, CAN, Pastry and Tapestry [69][64][65][86] propose distributed hash table based peer-to-peer lookup protocols, which utilize hash functions to map the resource requests into the related peer nodes. They provide a simple key-node mapping mechanism; however, most of them did not consider peer heterogeneity and any key locality properties. Catalog [24] proposes a peer-to-peer based catalog service to support queries in large-scale data sets. It is built on chord and focuses on finding the related node set associated with a query. PeerCQ [27] proposes a distributed hash table based peer-to-peer continual query system. PeerCQ also starts to consider peer heterogeneity and CQ locality.

Recently a few peer virtualization schemes [10][27][63][87] are proposed to address load balancing and peer heterogeneity. In [10][27], the number of virtual peers assigned to a node is proportional to its capacity. [63] proposes a scheme to adapt the number of virtual peers in a node according to its load situation. [87] proposes a peer virtualization scheme to utilize the proximity among peer nodes in order to reduce the overhead of transferring the load from one peer node to another peer node.

Super peer approach [83] separates super peers from simple peers so that super peer will have more responsibilities such as routing and computing. Nonetheless, this does not

solve load balancing among super peers, as there is capacity heterogeneity among super peers. The top-K algorithm [2] proposes a top-k query retrieval in a super-peer HyperCuP topology. But it does not have any connections with load balancing and it is designed for the specific topology.

One problem with these peer virtualization schemes is that they generate a much larger number of virtual peers than the number of physical peers. But the routing algorithm among peers still need to route the messages according to the virtual peers. The routing message may bounce back and forth among physical peers. This causes the unnecessary communication overhead in the routing process.

One of main tasks in load balancing is to identify where the extra load should be assigned. Previous load balancing schemes need to periodically exchange load and capacity information when there are no overload nodes. They also assume that there is a central pool that records the information about overloaded peers and lightweight peers. This is not always true. In addition, they ignore many factors including the cost for capacity in different nodes. In reality, the cost is different depending on the reputation and reliability of the nodes. PeerDS, described in Chapter 6, will address these problems.

2.5 Indexing Techniques in Subscription Based Systems

In [50][39], the authors study the filters applied in data streaming. But these filters are installed in the data sources and they mainly reduce the traffic between the data sources and the subscription server. Authors of [16][17] study the event filters between the subscription server and sinks. So their filters are similar to our filters. However, they did not try to do any optimizations for filter processing.

There is also work [46] that tries to use grouped filters to reduce the computation for each data source. Authors in [20] focused on schema based clustering for filters. Authors in [82] discuss index structures under Boolean Model.

Query grouping [7][46] and trigger grouping [71] group similar queries between the subscription server and data sources, remove the duplicate effort in the subscription server, and reduce the information traffic between the subscription server and data sources. However, in most subscription systems (e.g., the push interface in PeerDS), the subscriber cannot define its own query for a specific data source since there are too many data sources, or the subscription server wants to keep the data sources transparent to the subscribers. In this situation, the best way to reduce the information traffic between the subscription server and subscribers is to install filters customized by the subscribers. Filters between the subscription server and subscribers can also be combined with query grouping and trigger grouping to further reduce the information traffic.

The other problem with previous work is the scalability of filter management and processing. As the number of filters dramatically increases, the performance of filter processing will become the bottleneck. As there are many updates arriving at the same time, we need to schedule the updates being sent to subscribers. Without a proper filter management system, it takes a long time to find the real interests of each update. It is essential to find an efficient way for filter processing. Filter indexing proposed in Chapter 7 will address these problems.

CHAPTER 3 SYSTEM ON MOBILE DEVICES (SyD): AN OVERVIEW

This chapter gives the overview of SyD (System on Mobile Devices). Our research has been conducted within the SyD project; hence, it is essential for the readers to understand the SyD middleware. The quality aware transaction processing framework proposed in Chapter 4 is the extension of SyD to support distributed transactions. PeerDS, the directory service described in Chapter 6, is the extension of SyD directory service. All collaborative applications introduced in this research are developed in the context of SyD. The SyD project has been led by Prof. Sushil Prasad of Georgia State University with Prof. Shamkant Navathe and Prof. Vijay Madisetti from Georgia Tech as collaborators and has included a large number of graduate students from Georgia State University and Georgia Tech.

3.1 Background

There is an emerging need for a comprehensive middleware technology to enable development and deployment of collaborative distributed applications over a collection of mobile (as well as wired) devices. This has been identified earlier as one of the key research challenges [14][51]. Our work is an ongoing effort to address this challenge. We seek to enable group applications over a collection of heterogeneous, autonomous, and mobile data stores, interconnected through wired or wireless networks of various

characteristics, and running on devices of varying capabilities (pagers, cell phones, PDAs, PCs, etc.). The key requirements for such a middleware platform are to allow:

- A Uniform Connected View: Present a uniform view of device, data and network to ease programmer's burden. Provide a device-independent and a persistent (always connected) object-view of data and services, so as to mask mobility and heterogeneity. Allow multiple data models and representations on device data stores.
- Distributed Server Applications on Small Devices: Enable developing and deploying distributed server applications possibly hosted on mobile devices. Support atomic transactions across multiple, independent, and heterogeneous device-applications.
- High-Level Development and Deployment Environment: Enable rapid development of reliable and portable collaborative applications over heterogeneous devices, networks and data stores. Provide a general-purpose high-level programming environment that uses existing server applications and composes them to create possibly ad-hoc, yet integrated applications rapidly.

Limitations of Current Technology: The current technology for the development of such collaborative applications over a set of wired or wireless devices and networks has several limitations. It requires explicit and tedious programming on each kind of device, both for data access and for inter-device and inter-application communication. The application code is specific to the type of device, data format, and the network. The data store provides only a fixed set of services disallowing dynamic reconfiguration. Applications running across mobile devices become complex due to lack of persistence and weak connectivity. A few existing middlewares have addressed the stated

requirements in a piecemeal fashion. Limitations include only client-side programming on mobile devices, a restricted domain of applications. In addition, they are limited in supporting mobility, group collaboration and transactional properties.

One would think that a collaborative application that runs on a collection of heterogeneous possibly mobile devices can be easily developed with the existing middleware technologies such as JXTA, BREW, compact .NET and J2ME, but the reality is that they require many ad-hoc techniques as well as cumbersome and time-consuming programming such as remote procedure calls, transaction support, addressing heterogeneous hardware and software environment. Our System on Mobile Devices (SyD) middleware [60][59][57] has a modular architecture that makes such application development very systematic and streamlined. SyD is the first comprehensive working prototype of its kind, with a small footprint of 112 KB with 76 KB being device-resident, and has an excellent potential for incorporating many ideas in terms of functionality extensions and scalability.

3.2 Overview of SyD

System on Mobile Devices (SyD) is a new platform technology that addresses the key problems of heterogeneity of device, data format and network, and mobility. SyD combines ease of application development, mobility of code, application, data and users, independence from network and geographical location, and the scalability required of large enterprise applications concurrently with the small footprint required by hand held devices.

Each individual device in SyD may be a traditional database such as relational or object-oriented database, or an ad-hoc data store such as a flat file, an EXCEL worksheet or a list repository. These may be located on traditional computers, on personal digital assistants (PDAs) or even on devices such as a utility meter or a set-top box. These devices are assumed to be independent in that they do not share a global schema. The devices in SyD cooperate with each other to perform interesting tasks and we envision a new generation of applications to be built using the SyD framework.

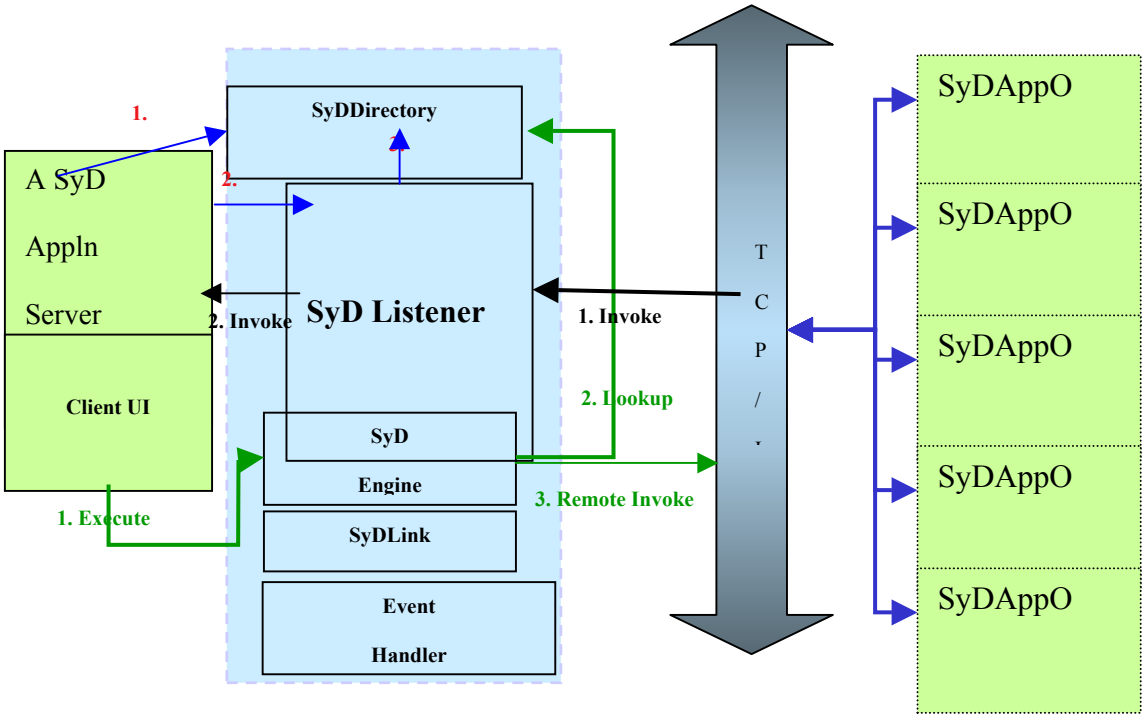
The SyD framework has three layers. At the lowest layer, individual data stores are represented by device objects that encapsulate methods/operations for access, and manipulation of this data. The SyD deviceware consists of a listener module to register objects and to execute local methods in response to remote invocations, and an engine module to invoke methods on remote objects. Object composition and execution of atomic transactions over multiple objects are provided by a bonding module. At the middle layer, there is SyD groupware, a logically coherent collection of services, APIs, and objects to facilitates the execution of application programs. Specifically, SyD groupware consists of a directory service module, group transactions and global event support, with application-level Quality of Service (QoS). At the highest level are the applications themselves. They rely only on these groupware and deviceware SyD services, and are independent of device, data and network. These applications include instantiations of server objects that are aggregations of the device objects and SyD middleware objects. The three-tier architecture of SyD enables applications to be

developed in a flexible manner without knowledge of device, database and network details.

The SyD groupware is responsible for making software applications (anywhere) aware of the named objects and their methods/services, executing these methods on behalf of applications, allowing the construction of SyD Application Objects (SyDAppOs) that are built on the device objects. SyD groupware provides the communications infrastructure between SyD Applications (SyDApps), in addition to providing QoS support services for SyDApps. SyDApps are applications written for the end users (human or machine) that operate on the SyDAppOs alone and are able to define their own services that utilize the SyDAppOs. The SyD groupware provides only a named device object for use by the SyDApps, without revealing the physical address, type or location of the information store. SyDApps are able to operate across multiple networks and multiple devices, relying on the middleware to provide the supporting services that translate the SyDApps code to the correct drivers, for both communications and computing. SyDApps can also decide on their own features and services they offer, without depending on individual databases residing on remote computing devices to offer those services. The SyD architecture is thus compatible with and extends the currently emerging Web services paradigm for Internet applications.

SyD uses the simple yet powerful idea of separating device and data store management from management of groups of users and/or data stores. Each device is managed by SyD deviceware that encapsulates it to present a uniform and persistent object view of the device data and methods. Groups of SyD devices are managed by the

SyD groupware that brokers all inter-device activities, and presents a uniform world-view to the SyD application to be developed and executed on. The SyD groupware hosts the application and other middleware objects, and provides a powerful set of services for directory and group management, and for performing group communication and other functionalities across multiple devices. SyD allows rapid development of a range of portable and reliable applications by hiding the heterogeneity and remoteness of data stores. Figure 3 shows the recently prototyped SyD kernel architecture.



(SyDAppO: SyD Application Object)

Figure 3. SyD Architecture

Some of the technologies employed by the current prototype include HP's iPAQ models 3600 and 3700 interconnected through IEEE 802.11 adapter cards and a 11 MB/s

Wireless LAN, TCP Sockets for remote method invocation and JAVA RMI for local method execution through reflection, MIDP/CLDC Java environment and XML standard for all inter-device communication.

Current applications include a system of calendars as well as a fleet management system on iPAQs. The ongoing and future work involves making SyD WSDL/SOAP compliant, developing an integrated development environment for SyD applications, porting SyD to devices other than iPAQs such as Palm Pilots and cell phones, obtaining a pure peer-to-peer version possibly leveraging off JXTA's directory service, providing more robust QoS functionalities including real-time constraints and application-level QoS, and addressing security issues.

3.3 Summary

System on Mobile Devices (SyD) which is the first working middleware prototype supporting an efficient collaborative application development for deployment on a collection of mobile devices. Our prototype also supports peer-to-peer and server applications. The existing middleware technologies such as JXTA, BREW, compact .Net and J2ME, provide a practical solution for supporting mobile applications involving PC's, PDA's and a host of devices. Yet, they require many ad-hoc techniques as well as cumbersome and time-consuming programming. One of the main advantages of SyD is a modular architecture that hides inherent heterogeneity among devices (their OS and languages), data stores (their format and access mechanism) and networks (protocols, wired or wireless) by presenting a uniform and persistent object view of mobile server applications and data-stores interacting through XML/SOAP requests and responses.

SyD has been driven by the current practical necessity to contribute fundamentally on certain aspects of middleware, database, Internet, mobile computing and related arenas. We briefly discuss SyD's place among the emerging middleware and distributed computing technologies, as well as where it is in terms of its goals. SyD as a middleware has a varying degree of flavors of object orientation (to mask heterogeneity), coordination orientation (for distributed coordination and ad-hoc group applications), and aspect orientation (distributed coordination among SyD objects with dynamic restructuring of embedded SyD bonds among coordinating objects). Additionally, it presents a reflection orientation with inspection, reference decoupling and dynamicity of groups through SyD Directory, and adaptation through smart proxy management, real-time tracking and scheduling of sub-invocations and application level QoS.

The overriding philosophy of SyD has been to be simple and modular, from middleware design and implementation to application development and execution. The main goal has been to mask heterogeneity of device, data, language/OS, network and mobility for rapid collaborative application development. These have been achieved to the following extent: (i) device and data store through listener, (ii) language and OS through generic remote invocation semantics, (iii) network by XML vocabulary for all inter-device communication, (iv) mobility through reference decoupling and replication/proxy through combined workings of SyD engine, directory service and registrar. SyD achieves temporal decoupling through asynchronous invocation with various remote invocation options for mobile clients, and provides an always connected object view through persistence/proxy mechanism for mobile hosts.

This dissertation contributes in the following areas to the SyD architecture:

(i) Transaction Processing: Support quality aware transaction processing. Chapter 4 and 5 describe the details of quality aware transaction processing.

(ii) Directory Service and lookup of resources in peer-to-peer operations over SyD. Chapter 6 describes PeerDS, a DHT based peer-to-peer directory service with push-pull hybrid interface. Chapter 7 supports the efficient filter processing in large-scale subscription service for a directory server. PeerDS described in Chapter 6 addresses the scalability of directory service at the architecture level. Filter indexing proposed in Chapter 7 addresses the scalability further at the resource access level, especially for resource objects over dynamically changing environments.

(iii) Load balancing among heterogeneous peers. Chapter 6 provides some optimizing algorithms for load balancing among heterogeneous peers.

CHAPTER 4 QUALITY AWARE TRANSACTION PROCESSING FRAMEWORK

Various mobile devices such as cell phones, PDAs and laptops form a heterogeneous computing platform for mobile computing. As we discussed in Section 1.2, mobile peer-to-peer systems bring challenges to the traditional transaction processing techniques.

4.1 Transaction Model

The existing transaction models [13][61][11][73][74][52]0 do not sufficiently capture the complexity and the execution requirements of mobile and Internet applications since most of the existing models are developed with a database-centric view [18]. Besides, it is also assumed that transaction components are to be executed over applications that are cooperatively developed. Another problem with current transaction models proposed in mobile systems is that they assume an infrastructure based mobile network and require the mobile support station (MSS). With mobile ad-hoc networks, this assumption is no longer valid. Furthermore, these models assume that the operations in mobile devices belong to the client. With the advance in wireless technologies, mobile devices can easily serve as servers. Researches need to put effort on designing models so that they will provide server functions to mobile devices. SyD discussed in Chapter 3 enables mobile devices to act as servers. When mobile devices could act as both a client and a server, they form a mobile peer-to-peer system. Currently we do not have any transaction support for mobile peer-to-peer systems.

Current transaction models [52][53][54] allow strong and weak states. Strong transaction state meets ACID guarantees as in traditional transactions. The weak transaction state relaxes ACID guarantees and usually does read-only operations or local read/write operations. Weak transaction state is a great success compared to previous models since it improves the availability and performance of the transaction service. However these two states do not fully express the heterogonous environment and do not provide enough flexibility to transaction processing in such environment. New transaction models are needed to address application/workflow frameworks that capture the collaborative yet independent, mobile and heterogeneous nature of application components. We argue that a quality aware transaction model should be able to assign a transaction to multiple states (at least two states) depending on the situation of network traffic, computing and power resources. A quality aware transaction should be able to gracefully adapt to a certain state according to the QoS agreement.

4.1.1 Multi-state Transaction Model

In traditional transaction models, transactions can only be globally active in its active phase. In the multi-state transaction model described in Figure 4, we divide transactions into multiple active states, which include globally active state, group active state and local active state. When the size of the group increases or decreases, group active state will also be promoted or demoted within the new group. A transaction in one active state can adapt to other active states. We also add a new state called “Tentatively Committed”. In the group active state or local active state, transactions could be tentatively committed. The transaction in the tentatively committed state can be recalled to the active states.

Finally, globally active transactions can commit or abort after certain conditions are satisfied or known. Demotion and promotion among states are decided by various constraints such as connectivity, energy and resource availability or real-time requirements.

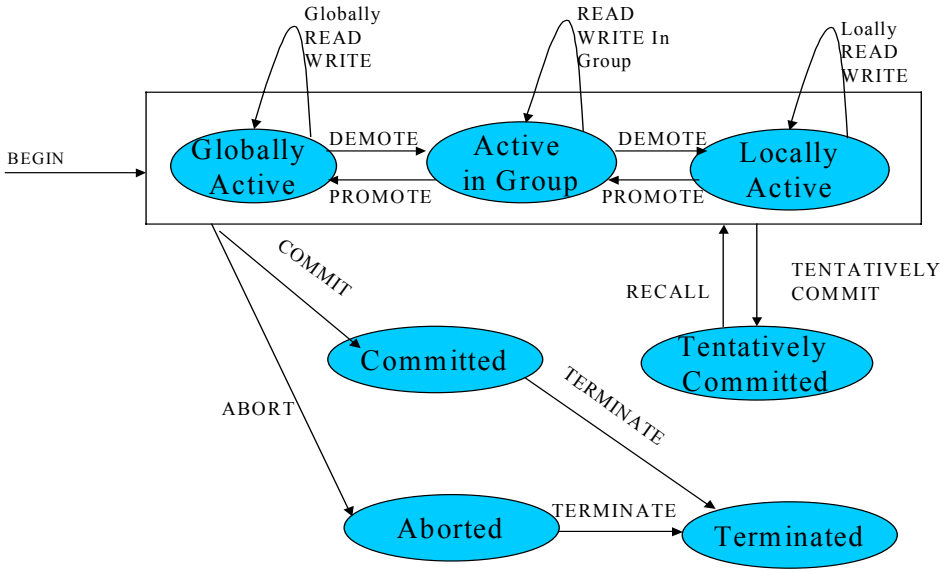


Figure 4. Multi-state transaction model

4.1.2 Quality of Service with Transactions

There are two different types of QoS associated with transactions: inter-transaction QoS and intra-transaction QoS. Inter-transaction QoS focuses on different quality requirements among transactions. Intra-transaction QoS focuses on quality requirements of individual transactions. The quality of a transaction is defined by how users "feel" about the transaction services.

Three categories can be defined for transactions with respect to real-time requirements:

- *Real hard transactions* that need to meet absolute deadline.
- *Hard transactions* that are between real hard transactions and soft transactions, which could miss some deadlines within a certain limit.
- *Soft transactions* that could miss deadlines.

These categories will be utilized in inter-transaction QoS control and can be used to set the priorities of transaction services. The metrics for measuring the quality of transaction service are listed as follows:

- *Correctness*: describes whether the response of the transaction service is correct.
- *Responsiveness*: describes the timeliness of the transaction service.
- *Availability*: describes whether the transaction service is available.
- *Information freshness*: describes if the response of the transaction service is up-to-date.
- *Security*: describes whether the transaction service is secure.

These metrics are utilized by intra-transaction QoS. A transaction state is defined by a set of these factors and represents a certain quality level of the transaction.

4.2 Adaptive Transaction

A transaction may adapt from the strongest transaction state to the weakest one based on changes in available resources. Developers of transaction services can define the transaction states between the strongest and the weakest transaction state, specify the

adaptation procedures and also predefine the control aspect as to when the adaptation occurs to meet the QoS requirements. In order to provide quality of service for distributed transactions, we need to relax the ACID properties that stand for Atomicity, Consistency, Isolation and Durability.

The motivation for relaxing ACID properties is to solve disconnection problems in mobile devices and to support long transactions. In mobile systems, we may never get all devices related to a transaction online at the same time. Such transactions would abort and never complete in traditional transaction processing. In this situation, relaxing ACID properties may be very useful. For instance, in a distributed calendar application, we may want to schedule a meeting among a number of users, say faculty members. If all user devices are connected, we can set up a group and schedule a global meeting; otherwise, based on some pre-defined policies, a sub-meeting among a subset of the group that are currently available can still be scheduled. When the disconnected devices are reconnected, the transactions can continue even if some formerly connected devices are disconnected.

Another motivation for relaxing ACID properties is to support cooperation among mobile devices in SyD. Consistency is mainly concerned with a serializable schedule. Atomicity is concerned with the size of the group related to the transactions in SyD. To further enhance the proposed transaction model, we add group serializability and group atomicity to maintain serializability and atomicity among groups. While we are relaxing ACID properties of mobile transactions, we use additional semantic information to guarantee the correctness of transactions and schedules.

Adaptive transactions could be divided into three categories:

- *Strong transactions* that maintains ACID properties globally.
- *Strong group transactions* that maintain ACID properties within groups.
- *Weak transactions* that maintain ACID properties locally.

4.2.1 Group collaboration

To facilitate group collaboration, we revisit and relax ACID concepts in transactions. We propose group ACID properties, which include group atomicity, group consistency, group isolation, group durability [77]. They provide a theoretic foundation for transaction adaptation. Based on them, we could provide much more flexible group collaboration than traditional transaction models.

A group in this model can be divided by time and space. For example, mobile devices may be disconnected at any time. At a certain time, connected mobile devices may form a group. After a while, some disconnected devices are reconnected and some connected devices are disconnected, a new connected group is formed. The common members in the two groups keep the state information of the transaction and will contact the new members in the second group to continue the transaction. On the other hand, devices may be in different cells or network partitions, which may be caused by some disasters or simply by power outage. There are no communications between these cells or network partitions. If a transaction needs to be executed in devices in these cells and partitions, the transaction can be carried from one partition to another partition by some moving devices.

4.2.2 A Sample Application: Distributed Calendar Application

To illustrate transaction adaptation problems, we use the distributed calendar application as our sample application. The architecture of distributed calendar is shown in Figure 5. A distributed calendar is a calendar application distributed among member devices that provides a shared global view. It can be viewed as a federated database. The calendar in a member device has two parts: a private calendar and a public calendar, which must be consistent. We will not address security issues in this chapter. All public calendars from member devices form the global view of the distributed calendar. To illustrate transaction adaptation process, we have designed four transactions T1, T2, T3 and T4 as follows. (R: Read, W: Write, Ci: the calendar of member i, r: conference room, p: projector, Si: Step i)

•Transaction T1: Schedule a meeting among all faculty members at College of Computing in Georgia Institute of Technology. T1 can be further divided into three steps.

–Step 1 (S1): Schedule a meeting among group members. The operation can be described as (R(Ci), W(Ci)) (1<=i<=N). It involves Read and Write operations on all calendars of all members.

–Step 2 (S2): Reserve the conference room. The operation can be described as (R(r), W(r)).

–Step 3 (S3): Reserve a projector (p) if there are no projectors in the conference room. The operation can be simplified as (R(r), R(p), W(p)).

•Transaction T2: Cancel an appointment in the calendar Ci of a member i. The operation can be described as (R(Ci), W(Ci)).

•Transaction T3: Cancel the reservation for the conference room. The operation can be described as (R(r), W(r)).

•Transaction T4: Cancel the reservation for the projector. The operation can be described as (R(p), W(p)).

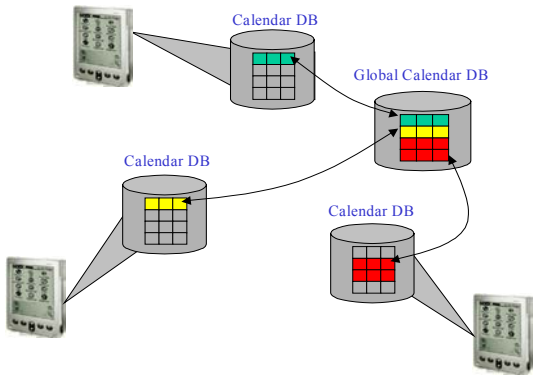


Figure 5. Distributed Calendar Application

4.2.3 Adaptive ACID Properties

In order to relax ACIP properties of transactions, we need to use additional semantic information to guarantee the correctness of the transactions. We focus on adaptive atomicity and adaptive consistency.

4.2.3.1 Adaptive Atomicity

Strong atomicity enforces that the whole transaction should be done either entirely or not at all. A transaction may be composed of component transactions. T1 includes S1, S2

and S3. S1 could be divided into sub-group schedule-meeting S1', S1'', S1''', … That is, we could represent T1 with the series of (S1', S1'', S1''', …), S2 and S3. In strong atomicity, failure of a component transaction of T1 such as S1'' means that all the work done by T1 has to be discarded. The failure of calendar application in one member device such as disconnection of the device could void other works done in all other member devices. In a large organization, the chance to get all member devices online at the same time is small. It would be very difficult or take a long time to complete such a transaction.

To improve the efficiency, we introduce *relative atomicity*. The operations at S1 in T1 can be described as R(C1), W(C1), R(C2), W(C2), …, R(Cn), W(Cn). As we could see from the semantic information, T3 and T4 could interleave within S1. T2 could not interleave within S1, but could interleave with S1, S2 and S3. T2 has different atomicity views of T1 from T3 and T4. *Group atomicity* is used to guarantee the atomicity of the transaction in the group. If a transaction has group atomicity, all other transactions have the same atomicity view of the transaction in the group. This feature is designed to facilitate the group collaboration in SyD.

If some mobile devices are disconnected, that means, some Cis are not available. If sub-groups are still connected, we adapt strong atomicity to group atomicity. The degree of group atomicity depends on the number of members in the group. When the size of a group changes, the degree of group atomicity varies. *Adaptive atomicity* is designed to adapt the atomicity of a transaction into different degrees.

4.2.3.2 Adaptive Consistency

Strong consistency enforces the same consistent view among members. Distributed calendar in all members has the same consistent view after the scheduling transaction. Similarly as relative atomicity, *Relative Consistency* is relaxed transaction consistency, which is allowed by semantic constraints.

Group Consistency is used to guarantee the consistency of the transaction in the group. If a transaction has group consistency, all other transactions have the same consistency view of the transaction in the group. For example, if some member devices are disconnected, we still schedule a meeting among connected members and sub-group members. The distributed calendar among connected members and disconnected members will then have an inconsistent view after the transaction. The inconsistent view will be conciliated after the disconnected members reconnect.

If some member devices are disconnected, that means, some Cis are not available. If sub-groups are still connected, we adapt strong consistency to group consistency. The degree of group consistency depends on the number of members in the group. Depending on the size of the group, there are different degrees of group consistency. *Adaptive Consistency* is designed to adapt the consistency of a transaction into different degrees.

4.3 QoS Architecture

To support quality aware transaction processing, we propose QoS architecture for the development and runtime support of quality aware transaction services (Figure 6). This is QoS extension of SyD. We now describe the main components in this architecture.

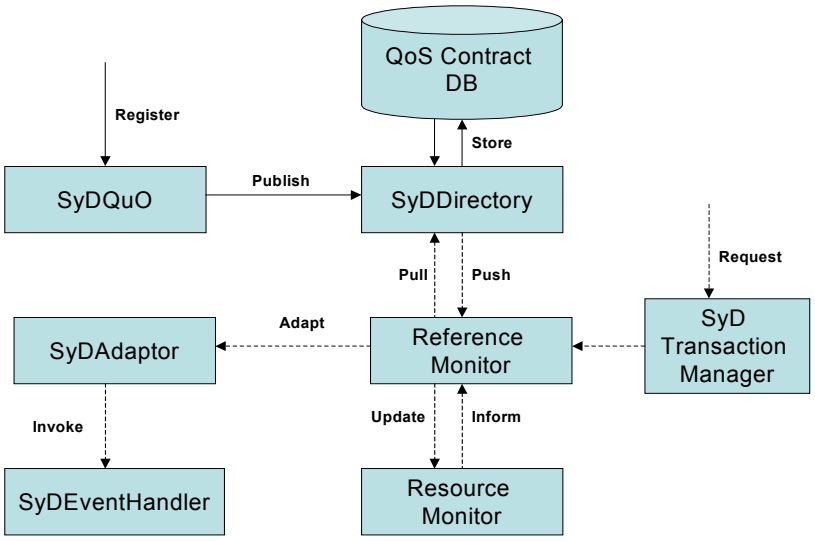


Figure 6. **QoS Architecture**

SyDQuO (SyD quality-of-service-object) is the core component in QoS extension of SyD. It describes and represents QoS contracts associated with transaction services. Each transaction service has the quality level, which may adapt to the transaction runtime environment. By default, the service level is the "best effort" level. *SyDDirectory* provides QoS contract publishing, management, and lookup services to SyD users and device objects. QoS Contract DB stores current available QoS contracts. *SyDEventHandler* handles local and global event registration, monitoring, and triggering. *Resource Monitor* monitors the changes of resources such as CPU, memory, storage, power, network bandwidth and disconnection. When the changes reach the predefined limit, it informs the reference monitor. *Reference Monitor* watches all valid QoS contracts of currently running transactions and sends the predefined resource thresholds

to the resource monitor. When the reference monitor receives any violations observed from the resource monitor, it checks the associated QoS contracts and calls the corresponding transaction adapter that invokes the adaptation procedures.

SyDAdapter manages the adaptation process of the transactions. Heuristic adaptation strategies could be provided. SyDAdapter supports intra-transaction QoS. SyDEvent is utilized to inform the services to adapt to the changes. SyD transaction manager is used to support inter-transaction QoS. It schedules the transactions according to their categories, deadlines, and waiting-time. SyD transaction manager can also maintains the quality level of transactions in different load situations based on feedback-based adaptations.

SyDQuo is responsible for the QoS contract of the transaction and the reference monitor enforces the execution of the QoS contract. In our framework, the QoS contract is separated from the mechanism to enforce it. This separation makes the development of QoS contract easier and the enforcement of the contract more secure. When a QoS contract violation is found, SyDAdapter is informed to execute the adaptation procedures defined in the QoS contract. SyDEventHandler can be utilized to process local and global events. Hence the transaction will adapt to a new transaction state. More details are available in [76][77][60].

4.4 Transaction Quality Description Language (TQDL)

As we discussed above, QoS specification is needed to develop QoS contracts and adaptation procedures in SyD. To support transaction adaptation, adaptive ACID properties need to be specified in the QoS specification. That means, we have to associate these relaxed ACID properties with QoS specification. Traditional generalized QoS

specification could not support this feature. So we design a transaction quality description language (TQDL) to specify QoS requirement and adaptation rules. A sample transaction QoS specification is shown in Figure 7.

```
<QoSSpec>
   <TranxQoS-Interface>
      <Name>DistributedCalendar.ScheduleMeeting</Name>
      <Group>All Faculty Members in College of Computing</Group>
      <Priority>High</Priority>
      <Event>
         <Name>Disconnection</Name>
            <Cond-Action>
               <Condition> Disconnection Rate = 0 </Condition>
               <Action> Enforce strong atomicity and strong consistency</Action>
            </Cond-Action>
            <Cond-Action>
               <Condition> 0<Disconnection Rate <1 </Condition>
               <Action>
                     Find connected members and sub-groups
                     Enforce relative atomicity and relative consistency in group
               </Action>
            </Cond-Action>
            <Cond-Action>
               <Condition> Disconnection Rate =1 </Condition>
               <Action> Enforce weak atomicity and weak consistency in local devices.</Action>
            </Cond-Action>
      </Event>
   </ TranxQoS-Interface >
</QoSSpec>
```

Figure 7. Sample transaction QoS Specification

TQDL is an XML-based extendable language. There are several motivations to adopt XML as the base of TQDL. First, TQDL has to be compatible with various heterogeneous environments in SyD. XML provides a perfect solution to this

heterogeneous requirement. Second, web service transactions are a new focus in SyD. By using XML, TQDL is a natural extension to Web Service Description Language (WSDL) and Web Service Transaction (WS-Transaction) Specification. Third, XML-based TQDL could use SOAP techniques to invoke adaptation procedure calls. Last, XML-based TQDL would be very easy to extend to a more complex language to provide additional functions.

Another feature of TQDL is to be compatible with Event-Condition-Action (ECA) rules in traditional databases. This would help database personnel to learn and develop transaction QoS specifications. It also provides a gateway in the future to convert legacy ECA rules into QoS specification.

4.4.1 Elements of a TQDL QoS specification

<TranxQoS-Interface> will define a QoS specification interface for a transaction. There are two main elements. The first element is to identify the transaction and the group and assign the priority level. <Name> describes the name of a transaction. <ID> describes the transaction ID. <Group> describes the group that the transaction belongs to. <Priority> defines the priority level of the transaction. The second element is to identify the events. The event in TQDL QoS specification mainly concerns resources such as power and disconnection. In an event, we add further constraints by identifying different conditions of the event. The conditions usually refer to the degree of resource changes. Actions are specified according to different conditions in the event. Actions usually refer to adaptation procedures. This Event-Condition-Action specification is similar to ECA rules in traditional active databases and refers to adaptation rules. <Event> describes the

events handled by the transaction. <Name> inside <Event> defines the event. <Condition> describes the different scenarios of an event. <Action> describes the actions that should be taken if the event occurs and the condition is met.

4.4.2 TQDL Code Generator

TQDL process and code generator is described as Figure 8. Transaction QoS specifications are TQDL documents. TQDL code generator reads and parses these transaction QoS specifications. Then it generates QoS contracts, adaptation rules and adaptation procedures. The reference monitor in SyD runtime monitors and enforces these QoS contracts and adaptation rules. The resource monitor monitors the changes in the resources according to events specified, and informs reference monitor the changes when the conditions specified are reached. Reference monitor calls adaptation procedures specified according to predefined adaptation rules.

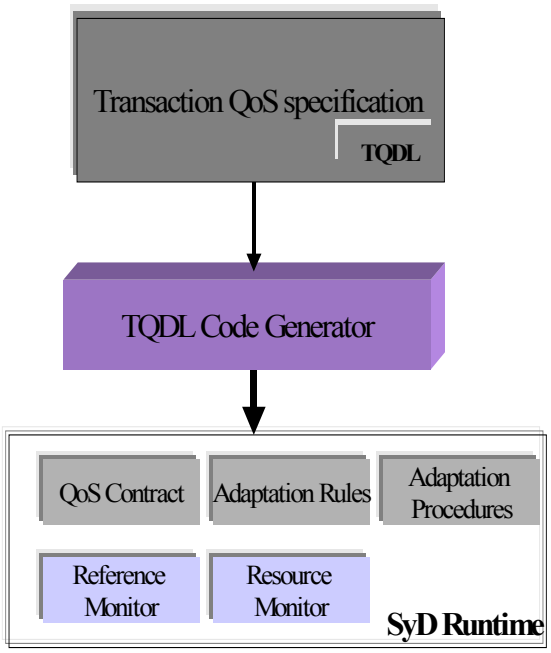


Figure 8. TQDL process and Code generator

4.5 Summary

Heterogeneous computing environment in mobile peer-to-peer systems poses a great challenge to traditional transaction processing techniques. Limitations of mobile peer-to-peer systems change the assumptions of traditional transaction processing and require new techniques for transaction processing.

Different transaction services have different quality requirements. The same transaction service needs different qualities in various resource availability situations. Quality aware transaction processing techniques extend QoS concepts and facilities from multimedia applications to transaction services. We provide a framework for

development and runtime support of quality aware transaction service and this framework supports flexible adaptation strategies for transactions.

In contrast to the weak transaction model proposed in most research works, we propose a multi-state transaction model that can adapt between the strongest and the weakest transaction model. A high quality transaction service can gracefully degrade to the low quality transaction service according to predefined QoS contracts. By systematically relaxing ACID properties, our techniques improve the availability and responsiveness of transaction services.

With the assumption of disconnection and weak resource availability, mobile transactions are long-lived. At the same time, mobile transactions need to support cooperation among mobile devices. In order to support long-lived and cooperative transactions, we relax ACID properties of mobile transactions. The quality aware transaction processing framework needs to use a specialized QoS specification language to describe the adaptation of ACID properties. Transaction Quality Description Language (TQDL) is developed to combine QoS specification with ACID properties. A distributed calendar application is used to illustrate adaptive ACID properties of transaction and associated QoS specification.

CHAPTER 5 PROBABILISTIC CONCURRENCY CONTROL AND TRANSACTION COMMIT PROTOCOLS

Based on the quality aware transaction processing framework in Chapter 4, this chapter proposes a probabilistic concurrency control mechanism and a group based transaction commit protocol.

5.1 System Model

Distributed transactions: One logical transaction may include multiple components, which execute at different sites. As we discussed before, this logical transaction could be a traditional database transaction or a collaborative application with transactional properties. A distributed transaction can be represented as T (P_1, P_2, …, P_n), in which P_1, P_2, …, P_n are participants of the transaction.

Transaction Group: There are two kinds of groups in a distributed transaction: logical group and physical group. Logical groups are formed on the basis of collaboration relationships among participants. Nevertheless, all participants belong to one group. There are sub-groups within the group. All these logical groups form the group hierarchy. Physical groups are formed on the basis of physical connectivity among participants. For the discussion of transaction commit protocols in this chapter, we will assume that a transaction group usually refers to a physical group.

Write-Set$_T$: the set of data items updated by the transaction T.

Read-Set$_T$: the set of data items read by the transaction T.

Based on the above definition, the system model could be defined as follows. There are N participants for a distributed transaction. There are M partitions among these N participants. A transaction could be represented as <V, G>. V is the vertex set and each vertex is a participant. G is the group set and each group represents a connected partition.

5.2 Probabilistic Concurrency Control

Current currency control mechanism utilizes locks to control the access to shared data. There are two kinds of locks: read lock (shared lock) and write lock (exclusive lock). Two locks from different transactions accessing to the same data are compatible if both locks could co-exist without compromising the serializability. The compatibility matrix in Table 1 demonstrates the compatibility between these access modes.

Table 1. The compatibility matrix for Access Modes

		Access Mode Requested	
		Read	Write
Access Mode Existed	Read	Yes	No
	Write	No	No

The two-phase locking protocol is the main concurrency control mechanism to enforce serializability and conflict-free. There are a variety of two-phase locking protocols depending on when locks are released. In all varieties of two-phase locking protocols, all lock requests precede all unlock requests for every transaction.

"Cascading abort" [6] occurs when abort of one transaction leads to abort of other transactions. In order to avoid cascading abort, usually all locks are released after the

transaction commits. (This is also called the strict two-phase locking protocol.) Hence, other transactions cannot access data items locked by uncommitted transactions. However, as we already know, the distributed transactions in mobile peer-to-peer systems are expected to be long-lived. So the shared data items would be locked and other transactions would be blocked for a long time if we adopt this protocol in mobile peer-to-peer systems.

In order to improve transaction throughput in the system, we need reduce the constraint of the two-phase lock protocol. At the same time, we want to control cascading aborts. We need to find a proper balance between long blocking and cascading aborts.

We propose a probabilistic approach to concurrency control. Instead of locks, we use references to record the shared access to data. A reference table is created to record the list of current transactions referencing each data item. Each mobile device has a reference table for the data items that it owns. All these referencing transactions could be regarded as staying at the second phase of two-phase locking protocols. A reference table described in Table 2 is an inverted index where each data item is followed by current transactions referencing the data item and transactions waiting for accessing the data item. Referencing transactions are ordered based on the order of accessing a data item. Waiting transactions are also ordered based on the order of initially waiting for the data item. Transaction commit probability is used to describe the possibility of a transaction's being committed.

Table 2. Reference Table

Data item	Ordered list of referencing transactions	Ordered list of waiting transactions
…	…	…

Probabilistic protocol: When a transaction T tries to access the data items in its write-set and read-set, the scheduler will check the reference table. If the access mode requested is compatible with the previous referencing access modes, the transaction T can access this data item and it will be added into the ordered list of referencing transactions. Otherwise, instead of blocking the transaction T in traditional concurrency control mechanism, the scheduler will compute the transaction commit probability $P_c(T)$ of the transaction T. If $P_c(T)$ is less than the probability threshold Pt, the transaction T will be blocked and added into the waiting list of the data item.

How to compute transaction commit probability of a transaction and how to set the probability threshold are two keys to the probabilistic approach. We will discuss the details in the following.

Figure 9 describes the dependency graph among transactions and data items. The commit probability of a transaction needs to consider two aspects: vertical aspect and horizontal aspect. The vertical aspect describes the commit probability based on the reference hierarchical levels of a single data item being accessed by a transaction. The horizontal aspect describes the commit probability based on multiple data items being accessed by a transaction.

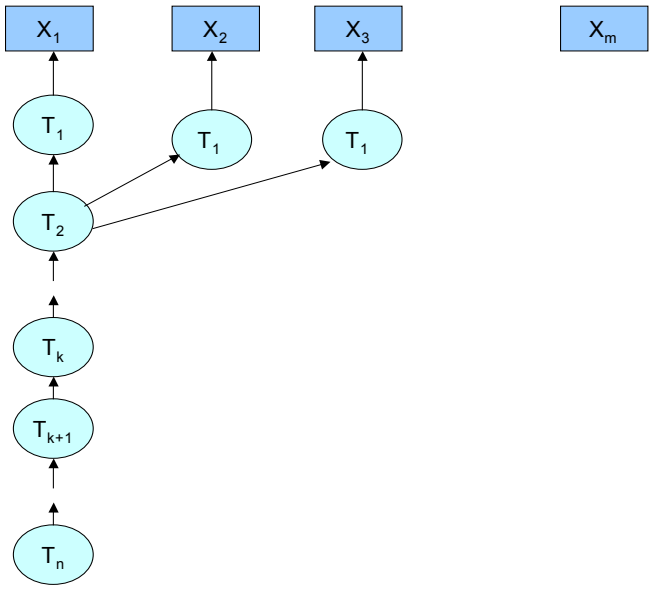


Figure 9. Dependency Graph

In the following, we discuss transaction commit probability based on the vertical aspect, i.e., probability based on a single data item. At the first level, a transaction can access this data item since no other transactions reference it. The commit probability of a transaction T,

$$P_c(T) = \frac{N_g}{N_t} \tag{5.1}$$

in which N_g is the number of transaction participants in the group and N_t is the total number of participants in this transaction. If the commit probability $P_c(T)$ of the transaction T > P_t , the data item could be accessed by the transaction T.

At the lower level, if the access modes are not compatible, whether we could grant the access to the data item is based on the resulting commit probability. Let X be the data

item being accessed. Let P_t be the probability threshold of the transaction T_{k+1} at level k+1. Let T_k be the transaction at level k. (The level stands for the referencing level of X.) The commit probability of T_{k+1},

$$P_c(T_{k+1}) = \frac{N_{g(k+1)}}{N_{t(k+1)}} * P_c(T_k) * \alpha \qquad (5.2)$$

$N_{g(k+1)}$ is the number of transaction participants in the group for T_{k+1}, $N_{t(k+1)}$ is the total number of participants in T_{k+1} and α is a reduction factor and $0<\alpha<1$. If the commit probability $P_c(T_{k+1})$ of the transaction $T_{k+1} > P_t$, the data item X referenced by T_k could be accessed by the transaction T_{k+1}.

What is the status of referencing transactions in the multi-state transaction model? Usually a referencing transaction is staying at the *tentatively committed* state. There is not enough information from all participants of the transaction to decide whether the transaction will finally commit. So they could commit or abort. When a referencing transaction at the top level commits, this transaction will be removed from the reference table. At the same time, we will adjust the status of transactions at the lower level and decide to commit them or continue to wait for more information. When a referencing transaction at the top level aborts, all transactions at the lower level will abort. This causes cascading abort. Fortunately, the degree of cascading abort is controlled. When a referencing transaction at the middle level decides to commit, it remains at the tentatively commit state. When its high level transaction commits, it will finally commit. When a referencing transaction at the middle level decides to abort, it will also abort the transactions that are at the lower level.

Now, let us look at the horizontal aspect. A transaction T may try to access multiple data items X_1, X_2, …, X_m. So we can compute multiple transaction commit probabilities $(P_c(T, X_1), P_c(T, X_2), ..., P_c(T, X_m))$ respectively for data items (X_1, X_2, …, X_m). The commit probability

$$P_c(T) = MIN\,(P_c(T, X_1), P_c(T, X_2), ..., P_c(T, X_m)) \qquad (5.3)$$

The other problem is to choose the proper probability threshold P_t. The choice of P_t decides the level of transaction commit ratio and the degree of cascading abort. Obviously, 0<= P_t <=1. When P_t increases, the possibility of a transaction being blocked increases and the degree of cascading abort decreases. We can adjust P_t based on the dynamic environment and the feedback from previous results.

When P_t = 0, transactions can access the data items anytime. There is no blocking. The serilizability may be maintained through other ways, however it will cause unlimited cascading abort.

Lemma 5.1: When P_t = 1, the probabilistic approach is a strict two-phase locking protocol.

Proof: From (5.1), we have $P_c(T) = \frac{N_g}{N_t}$.

When P_t = 1, so $P_c(T)$ >=1. We can deduce that $N_g = N_t$. That means all participants agrees to commit.

From (5.2), we have $P_c(T_{k+1}) = \frac{N_{g(k+1)}}{N_{t(k+1)}} * P_c(T_k) * \alpha$.

Since P_t=1, $P_c(T)$=1.

As 0<α<1 and $\frac{N_{g(k+1)}}{N_{t(k+1)}}$ <=1, we will have $P_c(T_{k+1})$<1. So $P_c(T_{k+1})$ will not be able to access the data items until $P_c(T_k)$ commits.

As the minimum commit probability is chosen to be the commit probability of a transaction when the transaction needs to access multiple data items. The conclusion still stands.

So the probabilistic approach is a strict two-phase locking protocol when the commit probability threshold is 1.

Another important problem for the probabilistic approach is to limit the effect of cascading abort. Lemma 5.2 gives the upper bound of the level of cascading abort.

Lemma 5.2*: The upper bound of the level of "cascading abort" is* $\frac{\ln(P_t)}{\ln(\alpha)}+1$.

Proof: From (5.2), we have $P_c(T_k)=\frac{N_{gk}}{N_{tk}} * P_c(T_{k-1}) * \alpha$.

Recursively applying (5.2) , we have

$$P_c(T_k)=\frac{N_{gk}}{N_{tk}} * \frac{N_{g(k-1)}}{N_{t(k-1)}} * \ldots * \frac{N_{g2}}{N_{t2}} * P_c(T_1) * \alpha^{(k-1)} \qquad (5.4)$$

From (5.1), we have $P_c(T_1)=\frac{N_{g1}}{N_{t1}}$

Replacing $P_c(T_1)$ in (5.4), we have

$$P_c(T_k)=\frac{N_{gk}}{N_{tk}} * \frac{N_{g(k-1)}}{N_{t(k-1)}} * \ldots * \frac{N_{g2}}{N_{t2}} * \frac{N_{g1}}{N_{t1}} * \alpha^{(k-1)} \qquad (5.5)$$

Let $\beta = \frac{N_{gk}}{N_{tk}} * \frac{N_{g(k-1)}}{N_{t(k-1)}} * \ldots * \frac{N_{g2}}{N_{t2}} * \frac{N_{g1}}{N_{t1}}$

Obviously β is a constant and $0 < \beta <= 1$

So we have $P_c(T_k) = \beta * \alpha^{(k-1)}$ (5.6)

As we know, a transaction can be allowed to access the data items only if it satisfies the following condition: ($P_c(T_k)$ is the minimum commit probability from multiple data items)

$$P_c(T_k) >= P_t$$

That means $\beta * \alpha^{(k-1)} >= P_t$

It could translate to $\alpha^{(k-1)} >= \frac{P_t}{\beta}$

Take logarithm at both sides, we have $(k-1) * \ln(\alpha) >= \ln(P_t) - \ln(\beta)$ (5.7)

Since $0 < \alpha < 1$, so $\ln(\alpha) < 0$

Divided $\ln(\alpha)$ from both sides in (5.7), we have

$$(k-1) <= \frac{\ln(P_t)}{\ln(\alpha)} - \frac{\ln(\beta)}{\ln(\alpha)}$$

This is equal to $k <= \frac{\ln(P_t)}{\ln(\alpha)} - \frac{\ln(\beta)}{\ln(\alpha)} + 1$

(5.8)

As $0 < P_t <= 1$ and $0 < \beta <= 1$, we know $\ln(\alpha) < 0$. So we can know

$$\frac{\ln(P_t)}{\ln(\alpha)} > 0 \quad \text{and} \quad \frac{\ln(\beta)}{\ln(\alpha)} > 0$$

From (5.8), we could deduce that

$$k <= \frac{\ln(P_t)}{\ln(\alpha)} + 1 \qquad (5.9)$$

So this gives the upper bound of the level of cascading abort.

Discussion: α is the reduction factor that is used to control the commit probability in the vertical aspect. As $0<\alpha<1$, a transaction T_2 in the lower level should have smaller commit probability than a transaction T_1 in the higher level as T_2 accesses the data item after T_2. When α increases, the priority that T_1 has over T_2 is smaller; the possibility of a transaction being blocked decreases and the degree of possibly cascading abort increases. So α is also called *Vertical Control Factor*. In contrast, P_t is called *Horizontal Control Factor* as it controls the threshold for the commit probability of a transaction accessing multiple data items. When P_t increases, the possibility of a transaction being blocked increases and the degree of cascading abort decreases.

5.3 A Group Based Transaction Commit Protocol

Although the transaction commit protocol is a well-researched database topic, it has been studied mostly in the context of distributed computing. The two-phase commit protocol [40] is the most widely used commit protocol. In the first phase, a coordinator polls all participants whether they want to commit or abort the transaction. In the second phase, the coordinator sends the decision to all components. Usually the decision is to commit if and only if all participants agree to commit.

In the traditional two-phase commit protocol and the three-phase commit protocol [68], they assume that there is a central commit coordinator and all participants of the transaction could communicate with this commit coordinator.

When these commit protocols moves to mobile and possibly ad-hoc computing, we find that frequent disconnections and partitions among transaction participants render current protocols ineffective and that no strategies exist for this setting yet. In turn, we devised a group based transaction commit protocol, which is based on our multi-state transaction model, in the presence of dynamic partitions.

A transaction process can be divided into two stages: execution and commit. So each participant of a transaction has these two stages. To simplify the discussion of the commit protocol in this section, we ignore the execution stage in the participant.

(**Definition 5.1**) *Synchronous Path*: A communication path existing between two sites that are located in the same partition at the same time.

(**Definition 5.2**) *Asynchronous Path*: If two sites do NOT have a synchronous path, but they are able to communicate through other sites passing the information between them at different time. Such a communication path is called an asynchronous path. Delay tolerant networks [21] describe the architectures for forming such an asynchronous path.

The assumptions for the group based commit protocol are described as follows:

i) Partitions can be detected.

ii) An asynchronous path exists between any two members.

iii) Each site maintains a log.

iv) A central coordinator is not always available.

v) Participants of a transaction are decided at the beginning of the transaction and will not be changed.

As we do not assume a central coordinator, we need to find a group coordinator for each group. The process of electing a group coordinator is described as follows

Step 1: Randomly pick any participants of the group

Step 2: Apply utility functions and find the optimal one. For example, the utility function could be the cost of communicating with other participants in the group.

The group based commit protocol is described as Table 3. The input is a distributed transaction with (n+m) participants. Network partitioning divides the transaction participants into different groups. We need to decide whether the transaction should be committed or aborted. In traditional commit protocols, the transaction will abort and restart later. If these participants are not connected together into a partition again, the transaction will keep aborting and not be able to complete. In the group based commit protocol, the transaction still could make the decision even if these participants are not connected into a partition again. The basic idea is to utilize the frequent membership change within different partitions. When a participant changes the membership from one group to another group, it could carry the transaction states in the old group into the new group. So the participants in the new group will know the decisions made in the old group so it will make the proper decision.

Table 3. Group Based Commit Protocol

Input: G(P1, P2, …, Pm, Q1, Q2,…, Qn)

Process:

Step 1: P1 initiates the transaction, P2, ..., Pm and Q1, ... Qn receive it. G is partitioned into two groups G1 (P1,..., Pm) and G2 (Q1,..., Qn). Let G1 and G2 elect the leader sites E1 and E2 for respective partitions.

Step 2: P1, P2… , Pm continue the process (Assume the process in each participant does not need the information from participants in the other partitions. For example, a query may be assigned to multiple distributed sites.). Similarly, Q1, … , Qn also continue the process. So G1 and G2 maintain group ACID properties and are in the group active state. If there are no conflicts within G1 or G2, G1 or G2 tentatively commits. Otherwise G1 or G2 aborts. Within each group, the two-phase commit protocol could be used to decide whether it should be tentatively committed or aborted.

Step 3: When the membership of G1 or G2 changes (e.g., some participants in G1 join G2 or some participants in G2 join G1, or G1 and G2 are merged into G'), we will know more information to make the decision whether the transaction should be kept tentatively committed, aborted or completely committed.

In this protocol, each site maintains a tentatively committed transaction queue. For each tentatively committed transaction, it stores the transaction ID, states (variables, corresponding values, constraints), and the associated group. Each site also maintains an aborted transaction queue. If the queue is too long, the participant will export it to the

disk. All these tentatively committed transactions and aborted transactions are recorded in the local transaction LOG.

Now let's revisit the *correctness criteria* for commit protocols proposed by Bernstein, Hadzilacos and Goodman. [4]

i) All processes that reach a decision, reach the same one.

ii) A process cannot reverse its decision after it has reached one.

iii) The commit decision can only be reached if all processes voted OK.

iv) If there are no failures and all processes voted OK, the decision will be to commit.

v) Given any execution schedule containing failures (of the type that the algorithm is designed to tolerate), if all failures are repaired and no new failures occur for a sufficiently long time, then all processes will eventually reach a decision.

As we can see from the group based commit protocol, a tentatively committed transaction could be promoted to being committed or demoted to being aborted. But a committed transaction will not be reversed. This satisfies Criterion ii). Based on Assumption ii) and the protocol, eventually all processes (in different groups) will reach the same decision. So this satisfies criterion i) and iii). An aborted transaction will abort the transaction from all participants. For Criterion iv), if there are no transaction state information exchanges among partitions, a transaction may be aborted after a long time even if each participant in every partition votes to commit. However, assumption ii) will make sure the asynchronous path among these partitions, so Criterion iv) is also satisfied.

Based on the same reason, Criterion v) will also be met. So the group based commit protocol will guarantee correct results.

If we want to extend the group based transaction commit protocol from abort veto mode to majority vote mode, the above correct criteria should be modified to base on majority processes instead of all processes.

5.4 Summary

Our probabilistic concurrency control mechanism provides a generalized approach to deal with long blocking while controlling the degree of "cascading abort". The commit probability threshold could be dynamically adjusted based on the feedback statistics from previous samples, changes of environment and expectation of concurrency level. The group based transaction commit protocol utilizes the multi-state transaction model to addresses frequent disconnections and network partitioning. The transaction state information in one partition could be carried over to another partition. A distributed transaction could still complete even if all participants could not be connected at the same time. Combining probabilistic concurrency control mechanism with the group based commit protocol, we could reduce the blocking and improve the transaction throughput and the transaction commit ratio.

CHAPTER 6 PeerDS: A SCALABLE, HYBRID PEER-TO-PEER DIRECTORY SERVICE

As we previously discussed in Chapter 1, mobile devices could and should be enabled to provide server functionalities. Co-ordination among applications requires a directory service that can provide a variety of functions. It brings the challenge to solve the scalability and dynamism for traditional directory service.

In order to improve scalability, we develop a peer-to-peer based directory service that is built on distributed hash tables. We designed an adaptive load-balancing scheme to reduce hotspots and distribute the load among the directory servers according to their load and capabilities. Distributed hash tables (DHTs) provide guarantees of an upper bound on the number of messages to find a key. There are a few DHT implementations in previous research. Our DHT implementation is based on Chord-like DHT design with additional mechanisms for *peer heterogeneity*, *resource locality*, and *routing speed-up* considerations.

In order to handle dynamism in the information, we develop a push interface in the directory servers besides the traditional pull interface. The push interface provides a mechanism to reduce the query load in the server by pushing information to the clients when the clients need real-time updates for resources. To a certain degree, it is similar to information monitoring and continual query processing. Different from other push interface solutions, our push interface is adaptive with the load or the changes of other requirements in the servers and in the clients.

Here is a motivating scenario for such a directory service application. In large-scale grid systems, a directory service is used to publish and lookup available resources in the system. A natural choice for a scalable directory service is a peer-to-peer architecture because an overwhelmed central directory service may cause the whole grid system to break down. A major problem faced in the peer-to-peer directory service is to find the result within a reasonable time limitation if the result is in the system. When a grid unit wants to monitor the resource updates, one way is to constantly send the requests to a directory server, the other way is to subscribe to the resource updates from the directory server. It can also setup filter functions to reduce the communication overhead. For example, it receives the update information only if the resource update exceeds a certain value.

6.1 System Architecture

PeerDS is a peer directory server that provides publishing, resource and group management to resource providers, and provides pull and push interfaces to clients. It is similar to the SyD directory component in Figure 3 and Figure 6. A resource hash table (RHT) is composed of keys, PeerDS nodes and summary of properties of the keys. A key is a hashed value of the name of a resource object, its group and its category. Properties of the key provide the functional and/or non-functional description of the service, resource or group associated with the key. The routing table (DHT) in each PeerDS node keeps track of a subset of all PeerDS nodes. The DHTs provide the routing among PeerDSs. All PeerDSs form a PeerDS ring as depicted in Figure 10.

A PeerDS node has a routing table, a successor node set and the predecessor for each peer identifier associated with the node. The node also stores part of the global directory database. In a PeerDS node, there are four interfaces: a publish interface that provides publish functionality for resource providers; a pull interface that provides regular lookup operations for directory clients; a push interface that provides subscription services to directory clients; and a peer network interface that supports communication among PeerDS nodes such as routing.

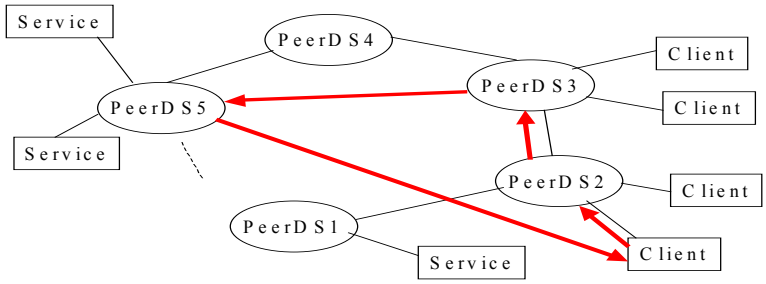


Figure 10. PeerDS system Architecture

6.2 Hybrid interface

The hybrid interface of directory service is described in Figure 11. A pull interface is typically provided in the regular directory services. Usually a client sends a lookup request to the directory server. After the directory server looks up the directory database, it sends the result back to the client. When a client wants to know the real-time updates of a resource such as location, it will frequently send requests to the directory server to avoid missing some important updates. If millions of clients choose to do this, as this is often the case in some important events such as Super Bowl games, the directory server

will be easily overloaded. Based on this observation, we could provide a push interface to push this information to the interested clients. A client can subscribe to the information it is interested in and specify a filter function so that only useful information would be transmitted back to client. Since many clients may be interested in the same information, the directory server only needs to process once for these groups of clients. In this way, we could improve the scalability of the directory server and reduce the communication overhead between a directory server and its clients. If there are many mobile clients, we could also move some computing functions to the directory server to save the energy in the clients. For example, rather than the client receiving the number of bullets remaining with each soldier and then totaling the number, the directory server could summarize them and send only the sum to the client.

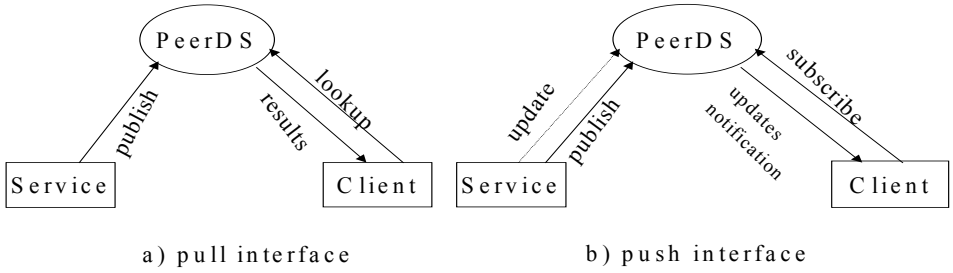


Figure 11. Hybrid interface

Figure 12 describes the scalability comparison between the push interface and the pull interface. The scalability is measured by considering the changes of load at the server with number of requests (clients) in both push and pull interfaces. The server load was measured by the response time perceived by the directory clients. Each lookup request to the pull interface requires a database operation that needs around 20ms. With the increase in the number of requests per second to the directory server, the server load in the push interface keeps almost constant since the server only needs one database operation for all subscribers to the same channel. However, the server load increases dramatically in the pull interface operations.

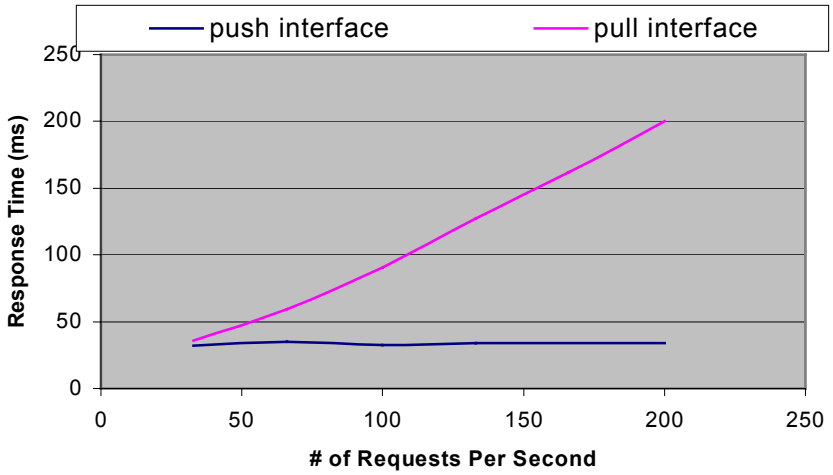


Figure 12. Scalability comparison between Push Interface and Pull Interface

6.3 Peer Heterogeneity and Load Balancing

6.3.1 Peer Heterogeneity and Virtualization

Typically peer nodes have different computing capabilities including CPU, memory, storage and network bandwidth. When we allocate the load, e.g., publish the resource's information, it would be natural to assign more load, e.g., publish larger number of resource objects, to the sites with more capabilities. As [34] mentioned, peer virtualization is one way to address peer heterogeneity. Existing approaches [10][27] assign a static number of virtual peer identifiers to a node according to the capabilities of the node when the node joins the overlay network. One of the problems in these approaches is, after we assign these virtual peers to a node, the node always keeps these virtual peers no matter how the capabilities and the load of the node changes. Intuitively, the number of virtual peer identifiers assigned to the node at the first time may not be appropriate. Furthermore, in a dynamic environment, the capabilities and the load of a node may change dramatically. [63] proposes a scheme to adapt the number of virtual peers in a node according to its load situation.

Our peer virtualization scheme [80] combines the adaptive approach proposed in [63] with node categories discussed in the following. In our adaptive design, the number of virtual peers assigned to a node dynamically changes depending on the load and capabilities of the node. We mainly focus on three capabilities: computing capability such as CPU and memory, storage capability such as disk space, and network capability including network bandwidth and traffic. We classify physical nodes based on computing, storage and network capabilities of nodes. When overload/hotspots occur, the

overloaded node sends the load balancing request so the number of virtual peers assigned to a physical node may dynamically change. When the load of a node reaches a certain threshold in one or all three capabilities, e.g. when the CPU load exceeds 90%, we should reduce the number of virtual peer identifiers associated with the node.

During load balancing, the system moves one or more virtual peers from the overloaded physical node to another physical node. When peer virtualization is more fine-grained, the average size of load allocated to each virtual peer is smaller. Therefore we have a more accurate estimate of the load being moved and the consumption of network bandwidth used to move load will be more efficient in this process.

One problem of these peer virtualization schemes is that they generate a much larger number of virtual peers than the number of physical peers. But the routing algorithm among peers still needs to route the messages according to the virtual peers. The routing message may bounce back and forth among physical peers. This causes unnecessary communication overhead in the routing process. The routing overhead is measured by $R_{overhead}$. As we know, $R_{overhead} = \frac{Hop_{Virtual}}{Hop_{baseline}}$, in which $Hop_{Virtual}$ is the average number of hops for a query in the virtualized P2Pnetwork and $Hop_{baseline}$ is the average number of hops for a query in the physical peer network. Figure 13 shows the routing overhead that peer virtualization causes. The number of physical nodes is set to 1,000. The baseline calculation is based on Chord [69]. We can see that the routing overhead dramatically increases with the total number of peers as peer virtualization generates a large number of virtual peers.

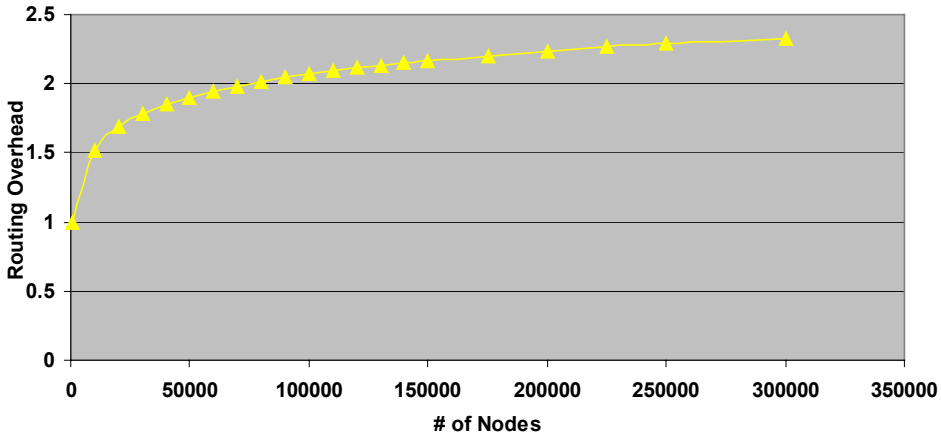


Figure 13. Routing overhead

On the other hand, in our best knowledge, all these schemes ignore the shared information among virtual peers in the same physical peer node. We argue that such shared information could be used to improve the routing among peers. For example, the routing tables (i.e. finger tables in Chord) and neighbor lists of all virtual peers residing at the same node could be searched to find the closer next node to the destination peer. Even if some existing schemes might inexplicitly utilize some of the shard information, which we do not see from their publications, they did not explore how the routing process is affected by the sharing, the degree of peer virtualization and the degree of peer heterogeneity. For example, if virtual peers residing at the same node share all information, could we get a better routing performance if the number of virtual peers residing at the same node increases as they will share more information and have a larger combined routing table?

Based on this information, we propose a new routing algorithm for the virtualized P2P network, which improves the average number of hops per query by 13% to 25% depending on the degree of peer heterogeneity and peer virtualization.

6.3.2 Load Balancing

We can represent the capacity of a node p as $\text{Capacity}_{\mathbf{p}}(c,s,b)$ and the load as $\text{Load}_{\mathbf{p}}(c,s,b)$. There is a load threshold $T_p(c, s, b)$. In the above terms, c denotes computing capacity/load/threshold, s denotes storage capacity/load/threshold and b denotes network bandwidth capacity/load/threshold respectively. If $\text{Load}_{\mathbf{p}}(c,s,b) > T_p(c, s, b)$, node p is overloaded. For the virtual peer i in the node p, the load is denoted as $\text{Load}_{\mathbf{p,i}}(c,s,b)$.

Adapting an overloaded node to normal condition or achieving a state where every physical node is lowered below the threshold is called load balancing. The specific implementation of adaptation is to utilize the algorithm for a virtual peer to join or leave the virtual P2P network. That is, removing a virtual peer is equal to a virtual peer identifier leaving the network, and adding a virtual peer is equal to a virtual peer identifier joining the network. The problems that we need to answer are therefore "*Which virtual peers should be moved*?" and "*Where should these virtual peers be moved to?*"

For the first problem, we want to find out the set of virtual peers in node p that minimize the load moved while satisfying the goal of reducing the load of overloaded node to normal. So we want to find out $(I_{p,1}, \ldots I_{p,n,})$ that minimizes $\sum_{i=1}^{n} Load_{p,i}(c,s,b)$ so that $\text{Load}_{\mathbf{p}}(c,s,b) < T_p(c, s, b)$ in which $I_{p,i}$ is the *i*th virtual peer in the node p.

For the second problem, we propose the top k peer selection algorithm to find the top k best nodes, which satisfy certain criteria, in the P2P network. “How to decide k?” is not the focus in this paper. But a rule of thumb is to choose k that equals to the number of virtual peers needed to be moved from p. We can get this number from the first answer to the first problem.

A top-k peer selection query includes two kinds of criteria: Selection Criteria (SC) and Optimization Criteria (OC). SC is the criteria that peers must meet. OC is the criteria that the query should optimize. A sample query is described informally as follows.

SELECT top k peers

Selection Criteria:

T(c,s,b) - Load(c,s,b) > mLoad(c,s,b)

AND Security_level > 0

AND Reliability >99%

AND Location = ‘USA’

AND Reputation >=’****’

AND Cost < $1,000

AND Distance < 500 miles

Optimization Criteria:

MIN(*Cost*)

In the above top k query, mLoad(c,s,b) is the additional capacity available to accommodate a virtual peer. So the peer should have enough spare capacity, the security

level should be above 0, the reliability should be above 99%, the peer is located at USA, the reputation should be equal or above 4 stars, the cost for mLoad(c,s,b) is less than the budget $1,000 and the distance to the overloaded node is less than 500 miles. We want to select top k peers that satisfy the above condition and have the cheapest cost per capacity unit. This is a cost-based load balancing scheme.

By using the optimization criteria in the above top k query, we essentially transfer our load balancing scheme into a multi-dimension multi-criteria decision problem. For example, if we want a proximity aware scheme, we could use MIN(Distance) as the optimization criterion. If we need a reputation-based scheme, we could use MAX(Reputation) as the optimization criterion.

6.4 System Model

Structured P2P systems such as DHT based P2P systems provide an upper bound on the number of messages so that they guarantee the answer if the result is in the P2P network. As we can see from Chord, CAN, Pastry, Tapestry, PeerCQ and Catalog [69][64][65][86] [27][24], this feature is based on the design of identifiers in the distributed hash tables. There are two identifiers in a virtualized P2P system: peer identifier and resource identifier. In this paper, we mainly focus on peer identifier and resource identifier. In order to map a resource identifier to a peer identifier, both identifiers are carefully designed in an m-bit identifier ring modulo 2^m, where m is a system parameter (m=24 in our study) and 2^m is the identifier space, so that a peer node can be identified when a resource identifier is known. (Resource object identifier and resource identifier could be seen as keyword in Chord.) The identifier ring is depicted in

Figure 14. A physical node could be associated with multiple virtual peer identifiers. P, P'', P''' are three physical nodes. Virtual peers P_1, P_2, P_3, P_4 and P_5 are located in node P. Virtual peers P_1', P_2', P_3', P_4' and P_5' are located in node P'. Virtual peers P_1'' and P_2'' are located in node P''.

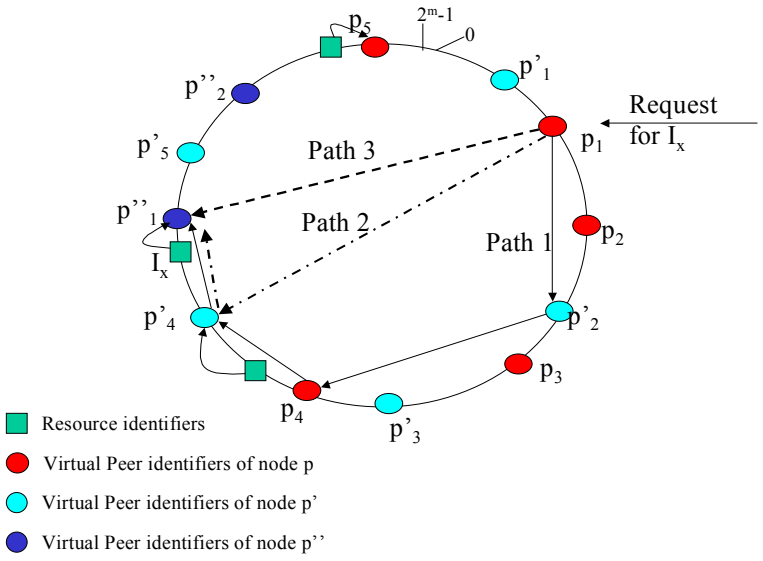


Figure 14. Identifier Ring

There are two layers of P2P networks in our system. One is the virtualized P2P network that we use to publish / lookup resource objects. The other is the physical network on which we maintain load balancing. Our top k peer selection algorithm is executed in the physical P2P network that has a much smaller number of peers than the virtualized P2P network. We use the new routing algorithm in the virtualized P2P network and Chord protocol in the physical one.

6.4.1 Peer Virtualization Data Model

Let Ip denote a peer identifier, Ir denote resource identifier, Np denote the number of peer identifiers, Nr denote the number of resource identifiers. Usually Nr >> Np.

Properties: Properties(Ip) of a peer identifier Ip describe the address, port, capabilities, node class and load information.

Predecessor: Predecessor(Ip) of a peer identifier Ip is the maximum peer identifier that is less than Ip in the peer identifier ring.

Successor Node Set SuccessorSet (Ip) of a virtual peer identifier Ip is defined as:

$$\text{SuccessorSet (Ip)} = \{ Ip_i \mid Ip_i = \text{SUCC\_NODE (Ip, i)} \ (0<=i<r)\}$$

in which SUCC_NODE (x, i) returns the ith minimum peer identifier which satisfies two conditions: i) it is greater than x in the peer identifier ring and ii) the node associated with this peer identifier x is different from the nodes that are already in the successor node set, and r is the number of successive nodes maintained.

Routing Table RoutingTable (Ip) of a peer identifier Ip for node p is defined as:

$\text{RoutingTable (Ip)} = \{ (Ipi, \text{Address}(Ipi)) \mid Ipi = \text{MIN\_NODE} ((Ip+2^{i-1}) \bmod 2^m) \ (0<=i<=Np)\}$ in which MIN_NODE (x) returns the minimum peer identifier which is greater than or equal to x in the peer identifier ring, and Address (x) returns the physical IP address of the peer identifier x.

Peer Identifier Descriptor: PeerDescriptor(Ip) of a peer identifier can be defined as

{Ip, Properties (Ip), Predecessor (Ip), SuccessorSet (Ip), RoutingTable (Ip)}

Node Descriptor NodeDescriptor(p) of a node p is defined as

$$\{(Ip_i, \text{PeerDescriptor}(Ip_i)) \mid Ip_i \text{ is one of virtual peers residing in node p.}\}$$

The comparisons in the above definition assume modulo 2^m operations. In the following discussion, all comparisons assume modulo 2^m operations unless otherwise specified. If peer p' is said to be closer to peer p than p'' is close to p, that means p' is in the clockwise path from p'' to p' in the identifier ring. (This is a very important point to understand the algorithms.) The routing node table is used for routing the information among virtual peer nodes. The successor node set is used for load balancing and fault tolerance. As we will discuss in the routing algorithm, the successor node set could also be utilized to speed up routing.

6.5 Algorithms

6.5.1 Routing Algorithm

In previous DHT protocols such as Chord, only the routing table of the peer identifier is used for the routing protocol. In our system, we utilize both the routing node table and successor node set of *all* peer identifiers in the node for the routing of a message. As we previously discussed, we assign varying numbers of virtual peers to a node according to the capabilities and the load of the node. A node p is associated with the node descriptor NodeDescriptor(p). The idea to speed up the routing process is to utilize the shared information in the node descriptor as the computation in the local node is much cheaper than the message communication among nodes. A typical situation of routing is to locate the proper peer identifier Ip given a resource identifier Ir. The algorithm to find the next peer identifier Ip' to which the request is forwarded from the current node p is described

in Table 4. FIND_CLOSEST_NODE () returns the peer identifier, which is the clockwise closest peer in the identifier ring to the destination peer.

Table 4. finding the next peer identifier (Routing algorithm)

```
0   Routing (Ir, Ip, p) {
1   If (Ir == Ip) return (Ip, p); // find the peer identifier and the node
2   Else {
3               (Ip', p') = (Ip, p) // initialize
4               For (i=0; i < GetNumberOfPeerIDs(p); i ++) {
5               // Find the closet peer identifier (Ip'', p'') in the routing table of this peer identifier
6                       (Ip'', p'') = FIND_CLOSEST_NODE (Ir, p.PeerIDs(i).RoutingTable);
7           // Find the closet peer identifier (Ip'', p'') in the successor node set
8           // of this peer identifier
9                   (Ip''', p''') =FIND_ CLOSEST_NODE(Ir, p.PeerIDs(i).SuccessorSet);
10              // Compare (Ip', p') with (Ip'', p'') and (Ip''', p''') to find the closet
11              // peer identifier in these three identifier pairs.
12          // Assign the closet peer identifier and its node to (Ip', p');
13              (Ip', p') = FIND_ CLOSEST_NODE (Ir, {(Ip', p'), (Ip'', p''), (Ip''',p''')});
14              }
15              return (Ip', p');
16  }
17  }
```

Throughout this chapter, the old routing algorithm refers to Chord routing algorithm in the virtualized P2P network. The new routing algorithm refers to our routing algorithm. The key difference between this routing algorithm and the old one is the loop from Line 4

to Line 14. The old routing algorithm will not search the routing tables and successor node sets of *all* virtual peer servers residing in the same node. This routing algorithm is not limited to our peer virtualization scheme. If a peer virtualization scheme does not maintain a successor node set for each virtual peer, the routing algorithm could ignore Line 9.

Now we use an example to compare the new routing algorithm and the old routing algorithm. In Figure 2, P, P' and p'' are three physical nodes. Each physical node may be allocated multiple virtual peers. P_1, P_2, P_3, P_4 and P_5 are virtual peers residing at the same node P. P_1', P_2', P_3', P_4' and P_5' are virtual peers residing at the same node P'. P''$_1$ and P''$_2$ are virtual peers residing at the same node P''. A query request for resource Ix is made to the virtual peer P_1. PATH 1 (P_1->P'$_2$->P_4->P'$_4$->P''$_1$) shows the routing path in the old routing algorithm, which only searches the routing table of the virtual peer. PATH 2 (P_1-> P'$_4$->P''$_1$) shows the routing path if we search all the routing tables of all virtual peers residing at the same node. PATH 3 (P1-> P''$_1$) shows the routing path of the new routing algorithm, which searches all the routing tables and successive Node Set of all virtual peer servers residing in the same node. As we can see from Figure 14, PATH 1 needs 4 hops to reach the destination, PATH 2 needs 2 hops to reach the destination and PATH 3 only needs one hop to reach the destination.

Hypothesis 1: given a resource identifier Ir, the routing in the virtual peer pp', which is closer to the destination virtual peer pp in the clockwise direction at the identifier ring than the other virtual peer pp'', takes equal or less number of messages to reach pp than the routing in a peer pp''. We formalize the hypothesis as follows:

Let PATH (pp', pp, Ir) be the routing path from p' to p and PATH (pp'', pp, Ir) be the routing path from pp'' to pp. Let Distance (pp', pp, Ir) be the number of hops of PATH(pp', pp, Ir) and Distance(pp'', pp, Ir) be the number of hops of PATH(pp'', pp, Ir).

If pp' is closer to pp than pp'' is close to pp in the clockwise direction, we have

Distance (pp', pp, Ir) <= Distance (pp'', pp, Ir)

The hypothesis is correct if pp' and pp'' are virtual peers that reside in the same physical node pp* since our routing algorithm will search routing tables and successor sets in pp* to find the same or closer next peer so that Distance (pp', pp, Ir) <= Distance (pp'', pp, Ir). Is it correct in all cases? We will discuss the details in Section 6.6.

Hypothesis 2: If virtual peers residing at the same node share all information, we get a better routing performance if the number of virtual peers residing at the same node increases as they will share more information and have a larger combined routing table. It is possibly better than Chord routing algorithm in the physical P2P network.

Intuitively it could be true. But we will use the simulation to verify part I of Hypothesis 2 and verify Part II in an analytical approach.

6.5.2 Lookup

Routing algorithm shows the routing procedure in a virtual peer. Lookup operation is the whole routing process that generates the routing path from the initiator peer to the target peer. The algorithm for a lookup operation posted to the peer node (Ip, p) is described in Table 5.

Table 5. lookup operation

```
0   Lookup (resource_category, resource_group, resource_object_name, resource_properties) {
1   // Generate the resource identifier based on the resource group
2   // and resource object name
3   Ir = GenerateResourceIdentifier (resource_category, resource_group,
4   resource_object_name);
5           (Ip', p') = Routing (Ir, Ip, p);
6           (Ip'', p'') = (Ip, p);
7           // Find the node that stores the information about the resource object.
8           While ((Ip', p') != (Ip, p)) {
9   Forward the lookup requests to node p'
10  Continue the lookup operation in node p'
11          (Ip",p") = (Ip', p');
12          (Ip', p') = Routing (Ir, Ip', p');
13  }
14  Now the node p' is the node that stores the information about the resource object.
15  Query the resource database in the node p' to return the records that satisfies resource
    properties including both functional properties and non-functional properties.
16  }
```

As we see from Table 5, Line 5 and Line 12 call routing algorithm to find the next node to be forwarded. Line 3 to Line 13 in the above lookup algorithm could be seen as the whole routing process that generates the routing path.

6.5.3 Top-K Peer Selection Algorithm

As we previously discussed, one of main problems in load balancing is to find the appropriate nodes to transfer the extra load. Previous approaches need to maintain metadata information about load and other information periodically in P2P system even if there are no overloaded nodes. Most importantly, they could not find out the best available peers, which satisfy the requirement of the overloaded node.

In order to find out top k physical under-loaded peers that optimize the design objective function, we need to have global information. Broadcast is one way to get this information. But regular broadcast approaches such as Gnutella [29] usually flood messages into the whole P2P network and it is uncertain when the query will end.

As we use the Chord protocol in the physical P2P network, we already maintain a finger (/routing) table in each physical node. So why cannot we reuse these routing tables to assist broadcasting?

Here we present a clever **broadcast** approach. The idea is to broadcast the messages to the nodes in the finger table. In the routing process, the message will be forwarded to only one node in its finger table. In the broadcast process, the message will be forwarded to multiple nodes in its finger table. The specific algorithm is presented in Table 6.

Table 6. Top-k Peer Selection Algorithm

Input: the overloaded node p, a Top-k query including Peer Selection Criteria (SC) and Optimization Criteria (OC),

Output: Top-k peers that satisfy SC

Phase I: Broadcast

1. p broadcasts the top k peer query into each node in its routing table.
2. Each node p' in the routing table receiving a top-k peer query will check itself to see whether it satisfy the requirement of peer selection criteria. If yes, it will add itself into its qualifying peer list.
3. P'' is the minimum identifier following p' in the chain of routing tables during the path from p to p'. p' will forward the top k peer query to the nodes in its routing table if these node identifiers fall into the identifier space between p' and p'' (clockwise). (Those nodes receiving the forwarded query start step 2.) If the number of queries p' forwards fno>0, and the number of responses p' receives rno<fno, p' will wait for the response from the nodes that it sends the query. Otherwise, p' enters the next phase.

Phase II: Aggregation

4. After p' collected all responses, it picks the top k peers from its qualifying peer list based on the selection criteria that we want to optimize. It could be cost, load, proximity, reputation or other criteria.
5. p' sends top k peers back to the source that it receives the request from.
6. p receives all responses from all nodes in its routing table, and picks the top k peers from all responses.

In this algorithm, the nodes select the top k peers based on the local information, peer selection criteria and optimization criteria in the query request. So the system does not need periodically exchange the dynamic metadata information such as load.

Theorem 6.1: Top-k peer selection algorithm takes exactly 2(N-1) messages with 2O(logN) hops. (N is the number of physical nodes.)

Proof. Let's first look at the broadcast phase. In this phase, every node other than p receives exactly one top-k query request. In aggregation phase, every node other than p sends a top-k query response back to the node that it receives the request from. So the algorithm will cost exactly 2(N-1) messages.

Based on the Chord protocol, we know that the maximum number of hops that p is needed to reach any other nodes is O(logN). As the maximum number of hops for p to receive a top-k result is also O(logN). The maximum number of hops for the algorithm is 2O(logN).

Theorem 6.2 (Correctness): Top-k peer selection algorithm selects the top k peers, which satisfy the Selection Criteria (SC) and Optimization Criteria (OC), in the whole P2P network.

Proof. The assumption here is that the attributes in selection criteria and optimization criteria do not change during the top-k peer selection. This is reasonable since the process needs only 2(logN) hops. As we select these top k peers from all peers, the aggregation phase guarantees that there are no better peers available to satisfy SC and OC. Under the above assumption, we know that the algorithm selects the best k peers.

6.6 Theoretical Analysis

In this section, we will focus on the theoretical analysis of our algorithms.

Claim 6.1: Chord routing algorithm in the physical P2P network can be decomposed into a knapsack problem.

Proof. In Chord, each node maintains, in the steady state, a table of nodes at a fixed distance from that node (more or less: nodes that are at least a fixed distance away). The distance from the start to the end node then becomes the knapsack; we want to fill the space provided with the lowest number of hops that we can.

Claim 6.2: Chord routing algorithm is optimal with its finger table in the physical P2P network.

Proof. Chord uses, at its base, a greedy algorithm, always taking as big steps as it can and as soon as it can. Since Chord's finger table consists of nodes that are distant by powers of 2, this algorithm always finds the optimal route for that finger table.

Now let us look at part II of **Hypothesis 2**. It could be rephrased into the following question.

Question 1: Could our routing algorithm in the virtualized P2P network possibly perform better than Chord routing algorithm in the physical P2P network as the number of virtual peer residing at the same physical node increases and they share more information?

Our routing algorithm is also a greedy algorithm always taking as big steps as it can at each step since Algorithm 1 picks the closest next node from the routing tables of all virtual peers. However, the hops we can make are dependent on the hops we've already

made, and the hop-sizes are no longer powers of two, so a greedy algorithm may not be optimal. Combining with the conclusion in Claim 2, the answer is "No".

Next we try to verify **Hypothesis 1**. Hypothesis I could be rephrased into the following question.

Question 2: Will the new routing algorithm that combines routing tables amongst virtual peers residing in the same physical peer, always perform at least as well as the Chord algorithm without sharing routing table combination?

The answer is "no". Here is a counter-example. Say there are virtual peers at addresses 0, 128, 130, 162, and 192 (and perhaps some nodes after 192). Say we start both algorithms at 0, and ask each one to find the virtual peer responsible for key 192.

The old algorithm (chord) would do the following. The key table for node 0 (and this can be found precisely, according the base algorithm for physical routing) contains only node 128 (and perhaps some nodes after 192). So from 0 it would make one hop to 128. Node 128 contains 130, 162 and 192, so it would make the next hop directly to 192. Total trip time is 2 hops.

Now, consider the new algorithm. Now, consider the case where the finger table for node 0 contains not only node 128, but also 130, due to peer virtualization. In this case, the algorithm will make a hop directly to node 130, since it is closer to 192. Now here's the tricky part. Node 130 is not guaranteed to contain anything but node 162, calculating by the way Chord maintains finger tables. So, it will make the jump to 162. Now, 162 knows about 192, so it will forward on the request. Total trip time is 3 hops.

So we can see that the new routing algorithm is not better than the previous algorithm in all cases, but we believe the average number of hops per query by using our new routing algorithm should be better than the old one. (We also verified this in the simulation section.)

In order to determine the number of hops taken by various key-based routing algorithms, it is necessary to build a mathematical model of the algorithm.

We start by making a number of simplifying assumptions. First, we assume that the number of active physical nodes $N=2^n$ for some positive integer n, and the address space is much larger than the number of physical or virtual nodes (which is a reasonable assumption, since Chord uses an address space of size 2^{160}). We further assume that the nodes are equally spaced within the address space. While this does not really reflect the actual state of affairs, it does allow us to examine the behavior of the system in the average case. This allows us to view the address space as continuous, but broken up into 2^n discrete, equal sections. While the sections will not be equal in reality, the random behavior of the algorithms allows us to make this approximation, since the more the number of nodes present in the system, the more it will approach this ideal.

When we introduce virtual nodes, we will make the assumption that the number of virtual nodes $M = 2^{m'}$ for some positive integer m'. The degree of peer virtualization (PVR) we represent as $\gamma = \frac{M}{N}$.

In order to measure the expected behavior of the system, we introduce a random variable X. X represents a random target in the address space that we will attempt to

locate using the algorithm (assuming a source location of 0, without loss of generality). We are interested then in H_X, the number of hops required to reach X from 0.

Theorem 6.3: In Chord, the expected number of hops required to reach a random target, $E(H_X)$, is $\frac{1}{2}n$.

Proof. In order to find this, we must count the number of possible targets for each potential $H_X = p$, which we call C(p). For p = 1, clearly, C(1) = n. More generally, C(p) = $\binom{n}{p}$. This number is difficult to analyze, but the fact that it is symmetrical around $\frac{1}{2}n$ gives an expected value for $p = H_X$, $E(H_X) = \frac{1}{2}n$. This result is also verified in the simulation results in Chord.

Lemma 6.1: In Chord with node virtualization, the expected number of hops,

$$E(H_X) = \frac{1}{2}m' = \frac{1}{2}(n + \log\gamma).$$

The proof of this result is similar to the proof above, since peer virtualization results in an identical number of hops as if each of the virtual nodes were a physical node.

For comparison, we define an algorithm like Chord, called Random. The difference is that, instead of maintaining a finger table of nodes at fixed distances, Random nodes maintain finger tables of size k of nodes at random points around the address space. At each step, if the target is not within its finger table, the algorithm picks a random target.

Theorem 6.4: In Random, the expected number of hops, $E(H_X)$, is $\frac{N}{k}$.

Proof. At each step of Random's trip around its address space, its probability of reaching its target is $\frac{k}{N}$, since the current node has a size k finger table, and there is a $\frac{1}{N}$ chance that each particular target is the correct one. This, therefore, is a straightforward Poisson distribution, and $E(H_X)$ is $\frac{N}{k}$.

6.7 Performance Evaluation

In this section, simulations are done to demonstrate and validate our design. We implemented a peer-to-peer simulator in C and a prototype in Java. The experiment data is a collection of keywords, which originally come from 80,000 HTML documents collected from 1,000 websites. The total number of different keywords is around 21,000,000. In addition, 160,000 search terms collected from a search engine are used to measure the performance of queries.

The hop length of the routing is the number of hops of the routing path of a query. It is the average hop length from many sample queries. In our simulation, we measure 160,000 queries to find the average hop length of the queries. In order to see the difference between queries where the results could be found in the peer-to-peer network or not, we measure the average hop length of FOUND queries, NOT_FOUND queries and total queries. A FOUND query is a query where we could find the matching results in the network while a NOT_FOUND query is a query where no matching results could be located in the network. In our simulation, there are 57,708 FOUND queries and 102,292 NOT_FOUND queries. For each query, we randomly choose a virtual peer as the initiator node. We measure the number of hops for each query and find the average number of

hops per query from 160,000 sample queries. Evaluating the average number of hops per query is the focus in our simulation.

Let R_{hop} denote the hop length ratio, HIR_{hop} be the hop improvement ratio, PVR be the peer virtualization ratio, PHR be the peer heterogeneity ratio. We have

$$R_{hop} = \frac{Hop_{old}}{Hop_{new}},$$

$$HIR_{hop} = R_{hop} - 1 = \frac{Hop_{old}}{Hop_{new}} - 1,$$

and $PVR = \frac{Total\ Number\ of\ Virtual\ \ Peers}{Total\ Number\ of\ \ Physical\ Peers}$

As we can see from the above formula, PVR will always be equal to or greater than 1.

In order to evaluate the impact of peer heterogeneity, we use three distributions for nodes: Uniform distribution, Gaussian distribution and Real distribution. In Uniform distribution, each peer will be assigned the same number of virtual peers. The number of virtual peers in a physical node is equal to Peer Virtualization Ratio (PVR). In Gaussian distribution, we use PVR as the mean and PVR/3 as the standard deviation in which the value of the standard deviation is based on the feature of Gaussian Distribution and the requirement of the simulation. Of course, the number of virtual peers in a physical node will always be greater than 0. Real distribution is the real world distribution that we deduced from the statistics of 1000 websites. It gives the probability of 0.1%, 1%, 0.6%, 1%, 4.2%, 7.6%, 17.4%, 33.5%, 22.6% and 12% for capacities of 900, 700, 450, 350, 250, 175, 125, 75, 30 and 1. Peer Heterogeneity Ratio (PHR) is the ratio to measure the degree that the system assigns the load according to their peer heterogeneity. This ratio is

between 0 and 1. If PHR = 0, the system views each node as being the same. This transfers into a uniform distribution with PVR=1. If PHR = 1, the system utilize the peer heterogeneity at the utmost degree.

In the simulation, we want to study the routing overhead, the average number of hops per query and routing improvement ratio with respect to peer virtualization ratio, peer heterogeneity ratio and the number of physical nodes. In addition, we want to verify Hypothesis 2.

As we mentioned before, the old (routing) algorithm refers to Chord routing algorithm in virtualized P2P network and the new algorithm refers to our routing algorithm described in Table 4 and 5.

6.7.1 Routing overhead comparison

As we previously discussed, routing overhead $R_{overhead} = \frac{Hop_{Virtual}}{Hop_{baseline}}$. Following up Figure 13, Figure 15(a) shows the routing overhead of two algorithms for 1000 physical nodes with Gaussian distribution. Routing overhead in both algorithms increases as PVR becomes larger since the total number of virtual peers (M=PVR*1000) also increases. We can see that the new routing algorithm reduces the routing overhead by 30% compared to the old one. As we previously explained, the routing message may bounce between two nodes in the old routing algorithm when we use virtual peers. The new routing algorithm removes the bouncing among peers and utilizes routing tables and successor sets of all virtual peers in the node to improve the routing. Nevertheless, the routing overhead is still high in the new algorithm. As $R_{overhead}$ is always greater than or equal to 1, there is

always routing overhead as long as there is peer virtualization. The baseline case could be seen as the optimal case.

Figure 16(a) shows the similar pattern for routing overhead of the two algorithms. The routing overhead increases as PHR increases since the total number of virtual peers increases becomes larger. The routing overhead in the new algorithm is improved by 0% to 27% compared to one in the old algorithm. This improvement increases as the PHR increases.

6.7.2 Comparison of Average number of Hops Per Query

We can deduce the average number of hops per query from our theoretical analysis. From Theorem 6.3, we can know physical theoretical value of the average number of hops per query in a P2P network

$$PTV = \frac{1}{2} LogN$$

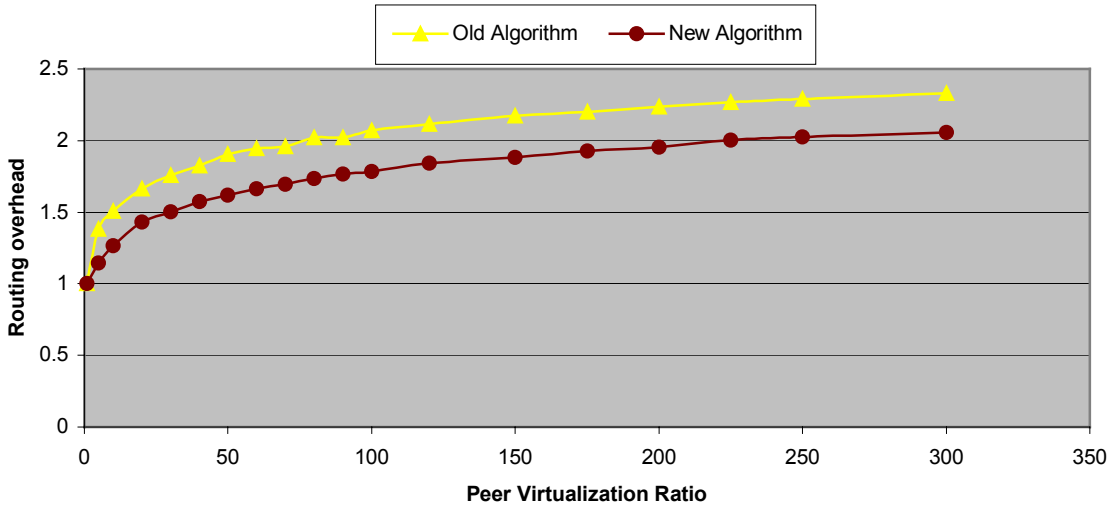
in which N is the number of physical nodes. From Lemma 1, we can know virtual theoretic value of the average number of hops per query in a P2P network

$$VTV = \frac{1}{2} M = \frac{1}{2}(LogN + LogPVR)$$

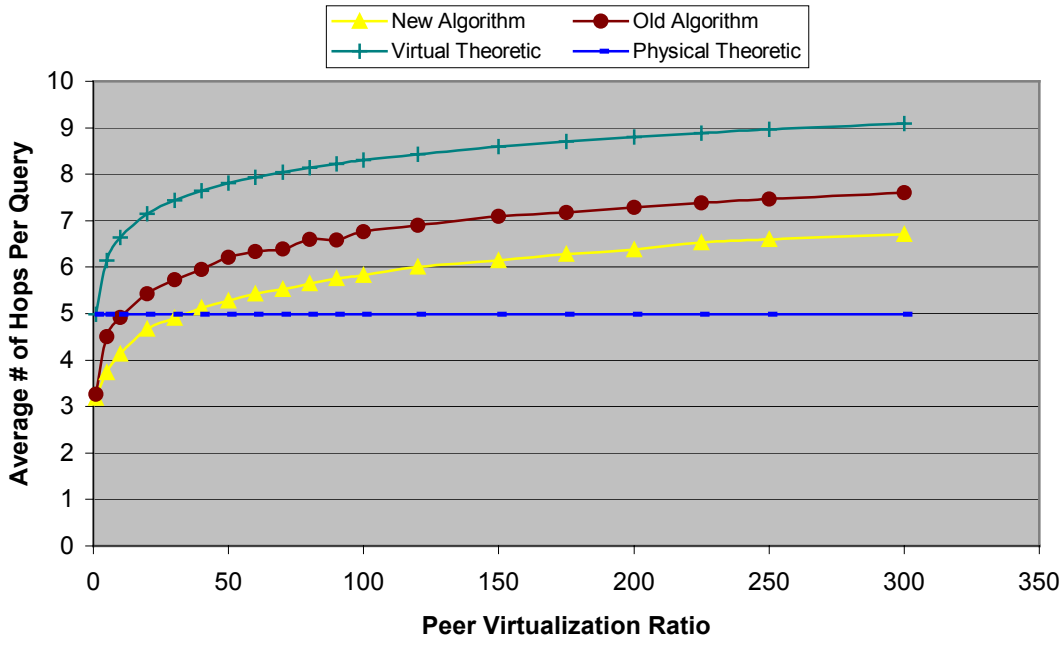
in which M is the number of virtual peers.

We now look at the average number of hops per query in the two algorithms. Figure 15(b) shows that the average number of hops increases when PVR increases. The average number of hops in the new algorithm is always equal or better than the one in the old algorithm. The two algorithms always outperform the virtual theoretical case. As the number of physical nodes is set to 1000, the physical theoretic value is a constant. After

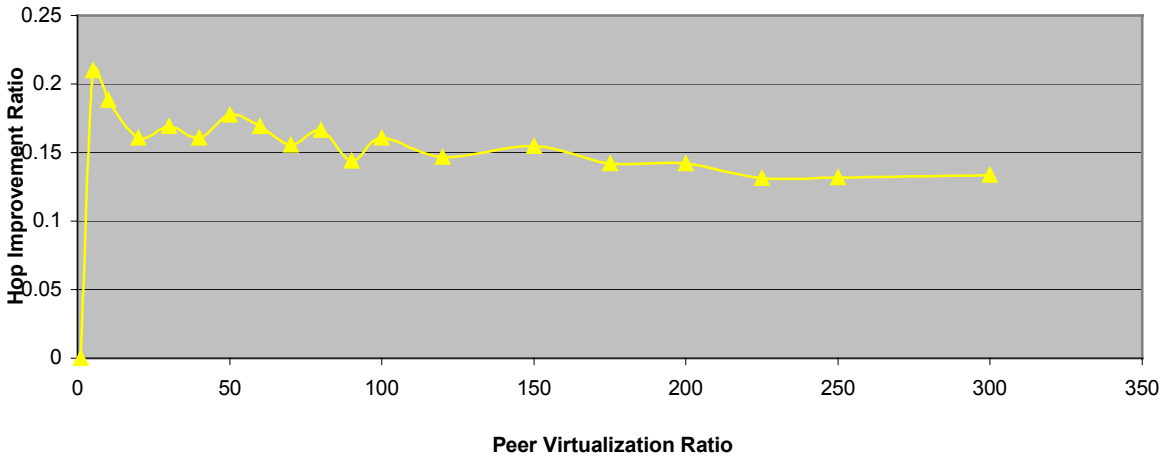
PVR>=10, the old algorithm underperforms the physical theoretical case. While the new algorithm trails the physical theoretical case after PVR>=40. So after PVR>=40, the average number of hops per query falls between the physical theoretic value and the virtual theoretical value. Figure 16(b) shows the similar pattern as Figure 15(b). As PHR changes from 0 to 1, the average number of hops increases in both algorithms and the new algorithm always outperforms the old algorithm.



(a) Routing overhead



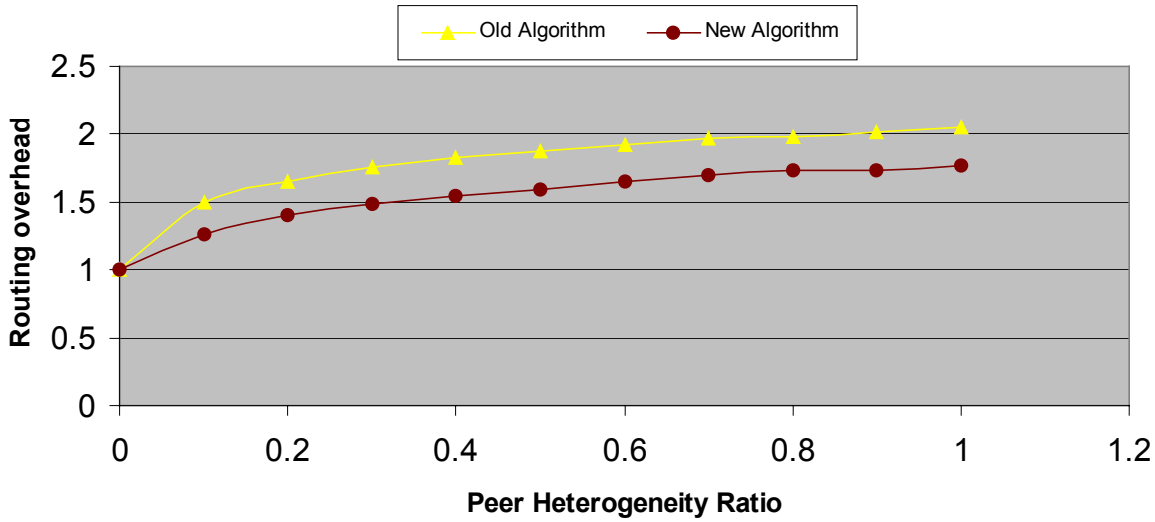
(b) Comparison of expected # of hops per query



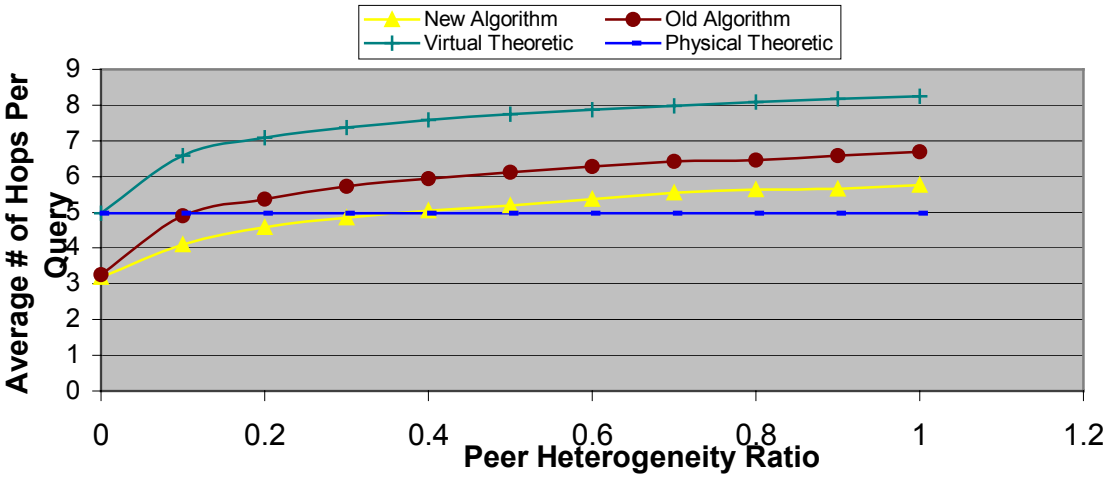
(c) Hop Improvement Ratio comparison

Figure 15. Impact of peer virtualization ratio

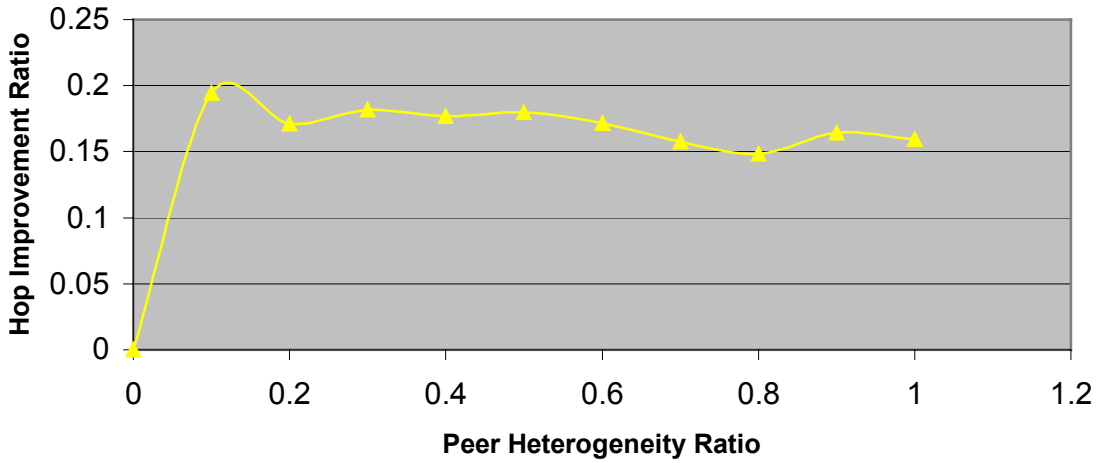
(Gaussian Distribution, # of Physical nodes =1000)



(a) Routing overhead



(b) Comparison of expected # of hops per query



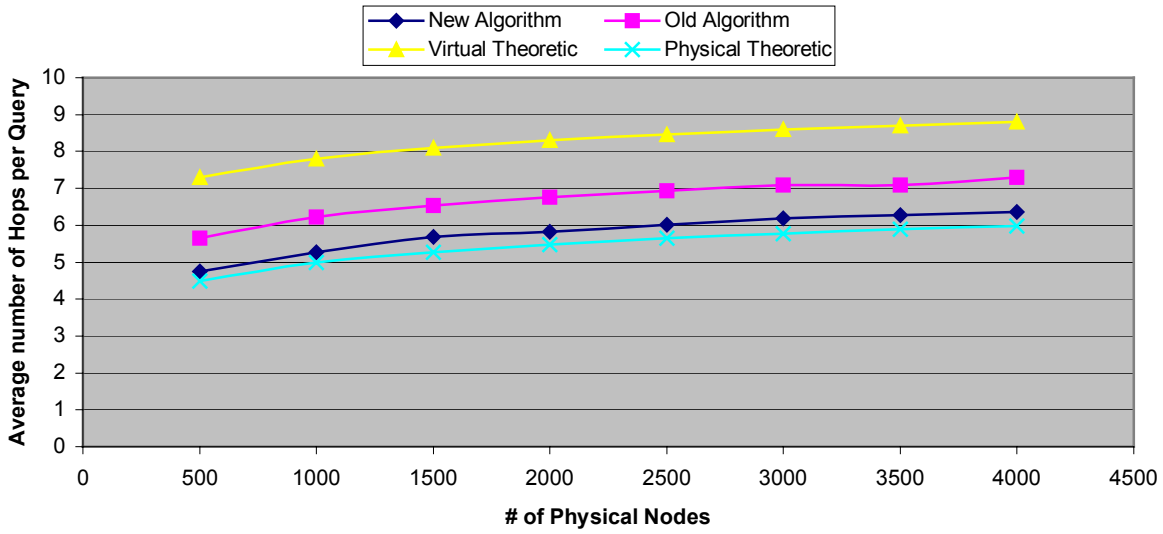
(c) Hop Improvement Ratio comparison

Figure 16. Impact of peer heterogeneity ratio

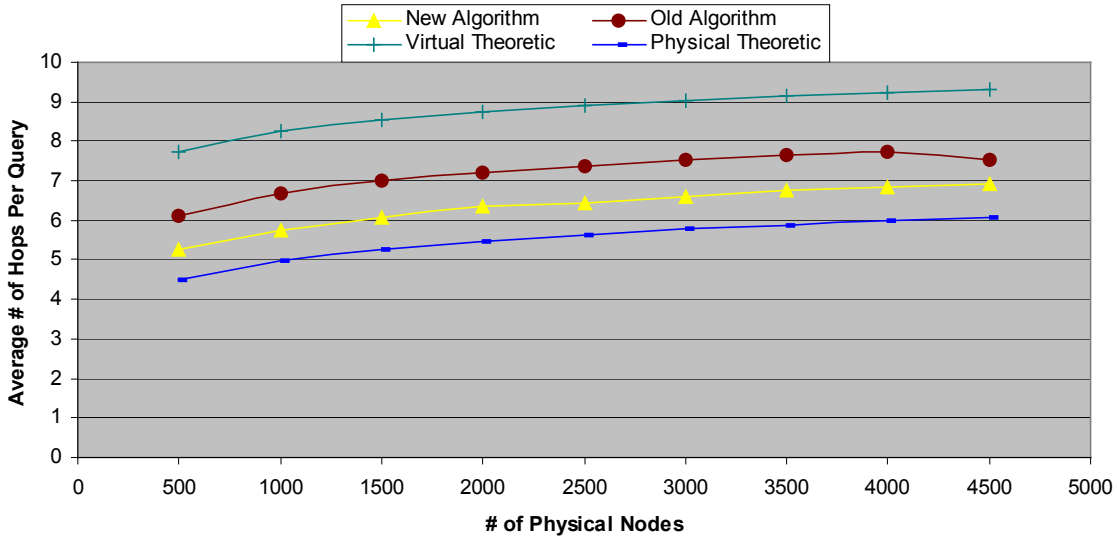
(Real Distribution, # of Physical nodes =1000)

6.7.3 Impact of peer virtualization ratio

Now we look at the impact of peer virtualization ratio. Figure 15(a) and (b) shows that the routing overhead and the average number of hops per query increase as PVR increases. This occurs as the number of virtual numbers dramatically increases. Figure 15(c) demonstrates that the new routing algorithm improves the average routing hop length of a query by 13% to 23% compared to the old algorithm. When PVR is smaller, the hop improvement ratio is better. As PVR increases, the total number of virtual peers increases but the size of each routing table does not change so the hop improvement ratio drops.



(a) Gaussian Distribution, PVR=50



(b) Real Distribution, PHR=1

Figure 17. Impact of the number of Physical Nodes

6.7.4 Impact of Peer Heterogeneity Ratio

Now we look at the impact of peer heterogeneity ratio. Figure 16(a) and (b) shows that the routing overhead and the average number of hops per query increase as PVR increases. This occurs as the increase of PHR incurs a larger number of virtual numbers. Figure 16(c) demonstrates that the new routing algorithm improves the average routing hop length of a query by 14% to 19% compared to the old algorithm. When PHR is smaller, the hop improvement ratio is better. As PHR increases, the total number of virtual peers increases but the size of each routing table does not change; so the hop improvement ratio drops.

6.7.5 Impact of the number of physical nodes

Figure 17 (a) shows that the performance of two algorithms lies between the physical theoretical case and the virtual theoretical case as PVR=50. As the number of physical nodes increases, the average number of hops in four situations also increases. The performance of the new algorithm is very close to the physical theoretical case. Figure 17(b) shows that the performance of two algorithms are between the physical theoretical case and the virtual theoretical case as PHR=1. As the number of physical nodes increases, the average number of hops in four situations also increases. In both figures, the new algorithm outperforms the old algorithm.

6.8 Summary

Structured peer-to-peer systems are popular solutions for large scale distributed computing and query processing. We implement a scalable peer-to-peer based directory

service called PeerDS, which is built on an improved distributed hashed table protocol. PeerDS supports both pull-based queries and push-based update multicasts to address dynamism, heterogeneity, complexity and scalability of information.

Heterogeneity among peers calls for peer virtualization to maintain a simple, yet powerful peer-to-peer overlay network. Nevertheless, peer virtualization generates a huge number of virtual peers and causes the unnecessary communication overhead in the routing process. In this paper, we propose a new peer-to-peer routing algorithm that reduces the number of hops of message forwarding and improves the performance of routing. We study the new and previous algorithms from the analytical perspective and through simulations. It shows that the average number of hops per query is improved by 15% to 25% in our algorithm.

In addition, we propose a Top-k peer selection algorithm for load balancing to find out the top k best available nodes in the P2P network with 2(N-1) messages within 2O(logN) hops. (N is the number of physical nodes.) The load balancing scheme is based on multiple factors which could be optimized on cost, proximity, reputation and other factors. This scheme eliminates the need to periodically maintain metadata for load balancing. And it does not need a central pool available to maintain load information of overloaded peers and lightweight peers. However, we do not claim that the Top-k peer selection algorithm could replace all other load balancing schemes. Rather it could be combined with other schemes to find a good match. For example, our scheme could be used when the overload/hotspots are not frequent. It provides a very useful mechanism to find the optimal k peers based on the design objective function.

CHAPTER 7 FILTER INDEXING: A SCALABLE SOLUTION

As we study the push interface (described in Figure 11 in Chapter 6), filters are installed in the directory server in order to reduce the information traffic and save the corresponding energy for subscribers in mobile devices. As we discussed in Section 1.4, filter processing in large subscription based systems may becomes the performance bottleneck. Furthermore, an additional question is: Which updates of resource objects should be sent first to maximize the system throughput (satisfying the requirements of the maximum number of subscribers for timely information)?

Our solution to the problem outlined above can be presented as a combination of two-pronged techniques: (1) Filter indexing: To speed up the filter process and remove the duplicate work from processing similar filters, we group and index filters from different subscribers for each resource object; (2) Scheduling based on filter index schemes. In this research, the emphasis of our work will be on the analysis of the first technique - indexing of filters.

7.1 Filter Indexing for a Single Attribute of a Resource Object

In this section, we discuss four indexing schemes for a single attribute of a resource object: Ad-hoc Indexing Scheme (AIS), Group Indexing Scheme (GIS), Group-Sort Indexing Scheme (GSIS) and B' Tree Indexing Scheme (BTIS).

7.1.1 Ad-hoc Indexing Scheme (AIS)

In the ad-hoc indexing scheme, filters are grouped by their associated resource objects. When an update of the resource object arrives, it will be compared with all filters associated with the resource object, even if some filters may be the same. This is a waste of computation as the number of subscribers who have the same filter increases when the subscribers grow. So we introduce a Group Indexing Scheme in next.

7.1.2 Group Indexing Scheme (GIS)

As we discussed in the AIS above, the same filters should be compared only once by an update of the resource object. All subscribers with the same filter will be added into the subscriber list of the filter. We call this as a Group Indexing Scheme. A variant of this scheme is to use a hash table. Since we need to compare whether the whole filter is the same, we could use the hash value of the filter to build a hash table. When a filter arrives at the subscription server, a hash function is applied to the filter. The hashed value will be used to find the entry of matching filters. However, in both situations, when an update of the resource object arrives, it still needs to be compared with all groups to find all subscribers who are interested in this update.

7.1.3 Group-Sort Indexing Scheme (GSIS)

Group-Sort Indexing Scheme is designed to avoid comparing with all groups of filters. In this scheme, filters are grouped by every operator on each attribute of each resource object. At the same time, filters are also sorted by the value of the attribute of the resource object, which is set in the filter. This creates linear indices. To simplify the

discussion, we only consider three basic data operators (>, <, =). So we have three indexing possibilities for each attribute of the resource object: greater group-sort index, less group-sort index and equal group-sort index. For greater group-sort index and equal group-sort index, the values of the attribute are sorted in *increasing* order. For less group-sort index, the values of the attribute are sorted by *decreasing* order. Every value in the index is associated with a list of subscribers that have the same filters with this value.

In addition, equal group-sort index can also use hash functions instead of sorting. We apply a hash function to the value of the attribute in the filter instead of the whole filter. When the updated value A' arrives, we apply hash function to A' to get the hashed value of A'. Then we look for this hashed value in the hash table. We return the associated subscriber list with the hashed value if we find the matching key in the hash table. Otherwise, the subscriber list will be empty.

When an update of the resource object arrives, three operator group-sort indices of each attribute should be searched. For equal group-sort index, we search for the updated value of the attribute in the index. If we find it, the associated subscriber list with this value will be returned. For greater group-sort index, we scan the index from the beginning until we find the updated value of the attribute or meet the key that is greater than the updated value. All subscriber lists associated with every value we scan before the end should be added into the returning subscriber list. Searching in less group-sort index is similar to searching in greater group-sort index.

Now, we merge three subscriber lists from three operator group indices of this attribute into the subscriber list for this attribute. Then we merge subscriber lists of all

attributes into the final subscriber list. This final subscriber list includes all subscribers that may be interested in this update of the resource object.

7.1.4 B' Tree Indexing Scheme (BTIS)

When the number of the filters increases, the overhead of searching the key in the group-sort indexing scheme also increases dramatically. We propose below a slight variation of B^+ tree called B' tree (pronounced B-prime tree) [78] and use an indexing scheme based on utilizing the B' tree. Similar to the group-sort indexing scheme (which uses linear tabular indexes), we build a B' tree for each operator for each attribute of the resource object.

B' tree is a variant of B+ tree. B' tree includes two kinds of nodes: internal node and leaf node. Data pointers are only stored in the leaf nodes. A data pointer points to the subscriber list associated with the key and the operator. The leaf nodes have an entry for every value of the attribute that is used with the operator in the filters, along with a data pointer to the subscriber list associated with the value of the attribute. The leaf nodes of B' tree are linked together to provide ordered access on the attribute to the subscriber lists. The leaf nodes linked list is almost the same as the index built in the group-sort indexing scheme. The main *difference* is that the leaf node linked list in less B' tree index is sorted in *increasing* order, whereas the less group-sort index sorts the linked list by *decreasing* order. The main difference between B' tree and B+ tree is that the B' tree maintains the number of subscribers for each subtree or each subscriber list in each node. This number will be used for scheduling of updates from different resource objects. At

the same time, we need to keep track of the leftmost leaf node, which will be used to retrieve the subscriber lists.

The filter processing of B' tree indexing scheme is similar to GSIS. Details of BTIS filter indexing process could be found in [78].

7.2 Filter Indexing for Multiple Conditions/Attributes of a Resource Object

Now we look at indexing the filters associated with multiple attributes of a resource object. Usually such a filter has multiple conditions connected by logical operators 'AND' and 'OR'. As we notice, multiple conditions/attributes in a filter will not affect AIS and GIS. So we mainly focus the discussion on GSIS and BTIS. 'OR' among conditions can be easily satisfied as the union of the subscriber lists from each condition forms the final subscriber list. 'AND' among conditions can be easily satisfied as the intersection of the subscriber lists from each condition forms the final subscriber list. However, since 'OR' and 'AND' coexist among conditions, we could not simply union or intersect the subscriber lists returned from each condition into the final subscriber list. As we can see, if a filter satisfies one condition of 'OR', the owners of the filter should be included in the subscriber list. However, if a filter satisfies one condition of 'AND', it does not necessarily mean that the owners of the filter should be included in the final subscriber list.

Based on this observation, we design the following scheme. First of all, we set a flag for all filters containing the 'AND' operator as these filters require additional process. When an update of the resource object arrives at the subscription server, we search the operator indices of all attributes of the resource object. For each search in an operator

index, a subscriber list will be returned. All subscriber lists returned from above searches are merged into two subscriber lists: the first list results from filter conditions without flags and the second list results from filter conditions with flags. In other words, the subscribers in the first list are generated from the single filter condition or the filter conditions containing the "OR" operator; the subscribers in the second list are generated from the filter conditions containing the "AND" operator. So we need to check the filters of these subscribers in the second list to make sure all conditions of the filters are satisfied. In this process, some subscribers in the second list may be thrown out from the list. After this, we merge these two lists into the final subscriber list.

7.3 Filter Indexing for Multiple Resource Objects

Usually a filter concerns one resource object. When a filter involves multiple resource objects, there usually are multiple conditions connected by logical operators 'OR'. If each 'OR' condition concerns a different resource object, we could break down the filter into multiple separate filters such that each filter concerns only one resource object. Then we could use the techniques in section 7.1 and 7.2 for each filter. If a filter does have the conditions related to multiple resource objects, we can use a similar technique used in section 7.2. At first, we union the subscriber lists returned by each condition; then we need to double-check the subscribers that have filters related to multiple resource objects. After this, we can get the final subscriber list.

7.4 Performance Evaluation

Experiments and simulations were done to study four indexing schemes. We implemented our prototype in Java SDK 1.4.1. We also built a simulator to simulate a large number of subscribers and filters to evaluate these indexing schemes.

The experiments are set up in two Dell Precision 360 desktops with Intel Pentium 4 2.8 GHz and 1GB Memory size. A subscription server is located in one desktop and a publisher is located in the other desktop. A 100Mbps Ethernet network connects the two machines.

Table 7. Experimental Parameters

Parameter	Value	Description
λ	5	Variable to control the frequency of updates
U1	100	Mean variable to generate values in filters
$\sigma 1$	40	Variance variable to generate values in filters
U2	90	Mean variable to generate values in updates
$\sigma 2$	50	Variance variable to generate values in updates
Sample size	150	The sample size to compute average search time

The publisher sends updates to the subscription server. Then the server will check the filter indices to find the list of matching filters. We use exponential distribution to control the frequencies of updates. Frequency f = log(r)/λ, in which r is a normal random variable and λ is the variable to control the frequency of updates. Normal distribution is

used to generate values of updates. Value V= u + σ * r, in which u is mean variable, σ is the variance variable and r is the normal random variable. The values of parameters are listed as Table 7 unless specified otherwise.

At first, we study the cost of building the index. In order to build indices for 200,000 filters, AIS needs 1,652ms, GIS needs 450,789ms, GSIS needs 407,916ms and BTIS needs 5,207ms. When the number of filters increases, the index building costs of GIS, GSIS and BTIS increase and the building cost of AIS remains flat. As the indices can be built when the subscription server starts, this cost is not the main issue of our concern.

7.4.1 Scalability

We study the scalability of four indexing schemes in this section. Experiments are run for four indexing schemes from 10,000 filters to 900,000 filters. Group sizes are not controlled. The values of filters are randomly generated so that the number of groups depends on randomness of the values in filters as is the case in real world. The group size is not uniform. *The search time measures the duration required by the subscription server to find the matching filters given an update.* The average search time is the mean of results from 150 updates. As we can see from Figure 18, the four indexing schemes are keeping the same bar in terms of performance until 600,000 filters are added. At 900,000 filters, the performance of AIS dramatically drops; GSIS is slightly better than GIS and BTIS. Surprisingly, BTIS does not outperform GSIS and GIS at 900,000 filters. The reason would be that only 65,000 groups are formed from 900,000 filters. So the impact of number of groups is the issue we want to address in the next section.

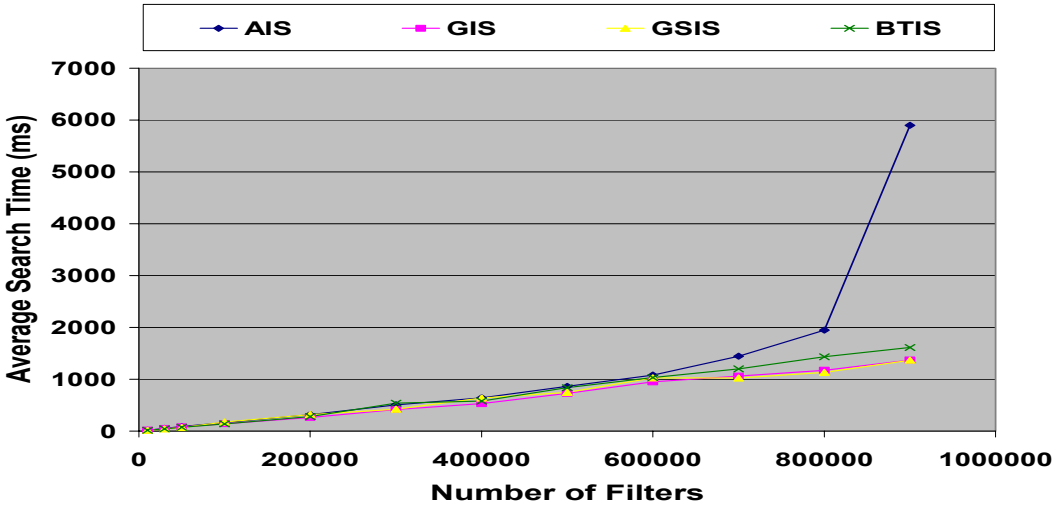


Figure 18. Scalability comparison of four schemes

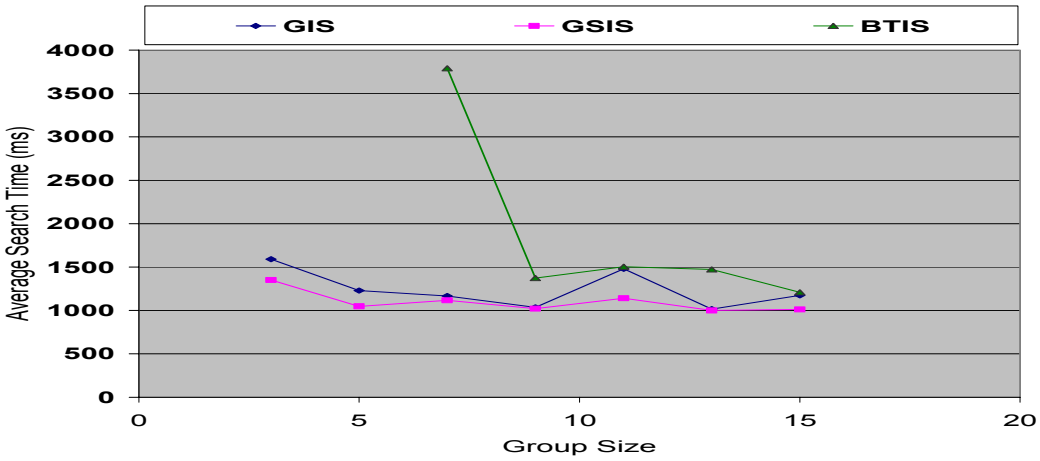


Figure 19. Impact of group size

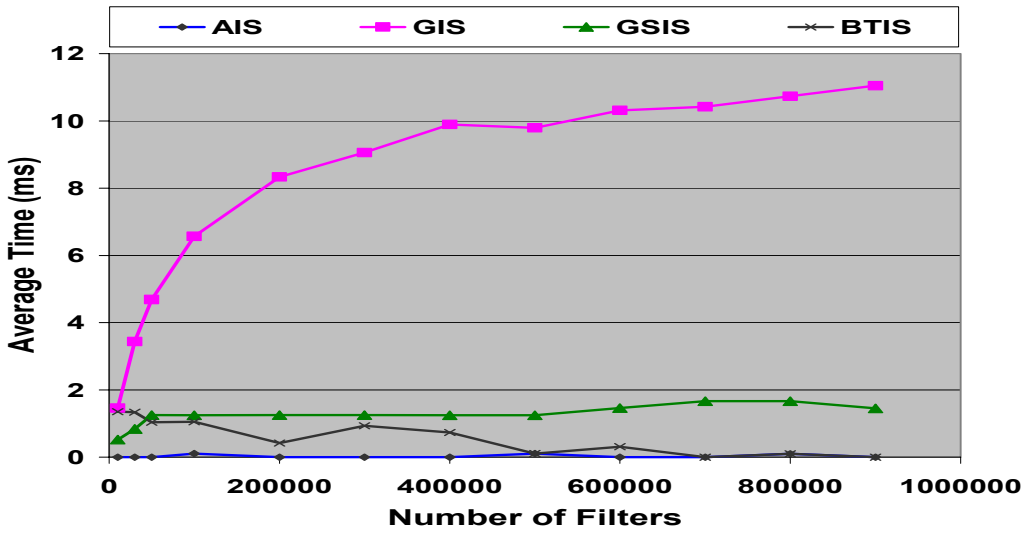


Figure 20. Performance comparison of adding a filter

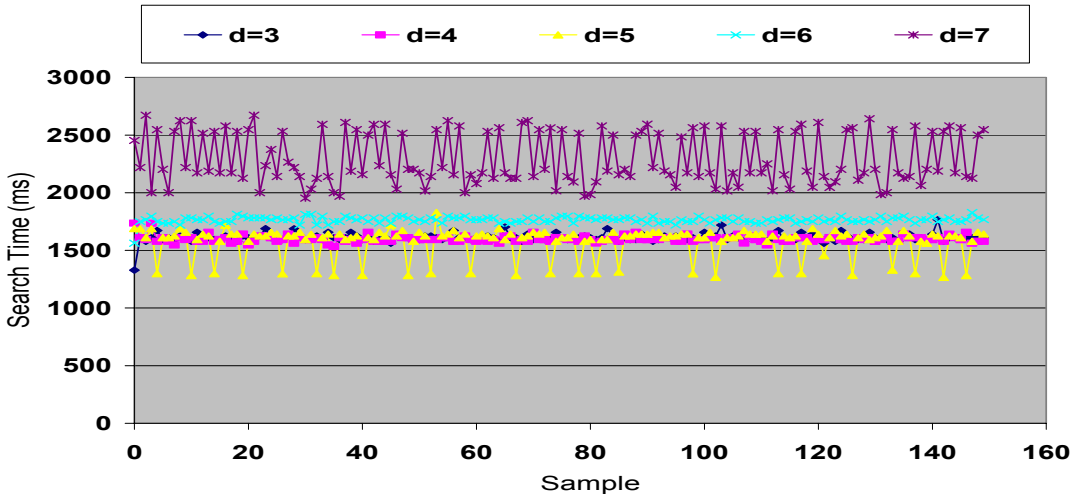


Figure 21. Impact of Degree of B' Tree

7.4.2 Impact of Group Size

To investigate the impact of group size and the number of groups, we set up the experiment with 800,000 filters. As we know, the number of groups * group size = 800,000. In Figure 19, we generate (800,000 / group_size) unique filters, then make as many copies as the group_size. As BTIS ran out of memory when the group size is 6, we only have the performance data from GIS and GSIS when group_size is less than 7. Surprisingly, GSIS always outperforms other two indexing schemes. BTIS's dramatic increase in average search time may come from memory contention when the group size is set as 7 and too many groups are generated.

7.4.3 Performance comparison of adding a new filter

We also study the performance of adding a new filter. We added a new filter when the number of total filters ranged from 10,000 to 900,000. We repeated this process 100 times and calculate the average cost for inserting a filter. Figure 20 shows that after 50,000 filters the overhead of inserting a filter causes a rapid increase in the average search time for GIS compared to other schemes. GIS needs a much higher overhead to insert a new filter when the number of filters is more than 50,000. The time for GSIS to insert a new filter is pretty stable at 1.5ms. The overhead of inserting a new filter in BTIS is very small and is insignificant when the number of filters exceeds 500,000.

7.4.4 Impact of the Degree of B' tree

We also study the impact of the degree of B' tree (block size in B' tree). The total number of filters is 900,000. In Figure 21, we show that the search time of 150 samples

for degrees ranging from 3 to 7. As we can see, the search time decreases when the degree changes from 3 to 5, then it increases when the degree from 5 to 7. We do not see a clear pattern as our data is limited.

7.5 Summary

Filtering of updates at the subscription servers is a popular approach to reduce information traffic in subscription-based systems, especially if most subscribers are located on mobile devices. Usually a subscriber sends its customized filter to the subscription management system and the system filters the information being sent to the subscriber. The filters are placed between the subscribers and the subscription server. With the increase of filters from subscribers, filtering may become a bottleneck and challenge the scalability of such systems.

In this chapter, we extend the push interface of PeerDS (SyD directory service) discussed in Chapter 6 into a general information subscription system. We show that filters could potentially reduce the information traffic between the subscription server and subscribers. However, the increasing number of filters may challenge the scalability of the subscription server. So we propose and investigate four filter indexing schemes: ad-hoc indexing scheme (AIS), group indexing scheme (GIS), group-sort indexing scheme (GSIS) and B' tree indexing scheme (BTIS). B' tree is proposed as a minor variation of the B+ tree. Experimental results show that GSIS is the most efficient indexing scheme for searching among the proposed schemes and BTIS has better performance for updating and inserting filters than others. BTIS can also help improving the scheduling of updates.

CHAPTER 8 FUTURE WORK

This chapter describes future work based on the ideas and solutions presented in this dissertation.

8.1 Architecture Extension For Dynamically Partitioned Network

As we discussed in Section 1.4.3, the partitions in dynamically partitioned network are constantly changing. That means, we do not have a stable partitioning of the mobile peer set. This brings the question: how could we continue and complete transactions in this dynamic environment? We are proposing various transaction processing architectures to investigate this problem.

8.1.1 Messenger approach

It is analogous to AOL, MSN and YAHOO messenger services which buffer and transfer the message when the receivers of the message are not online. In these commercial messenger systems, they use a central server to buffer the messages. It usually works well in normal environment but bears the typical drawback of the central servers. First of all, it is the bottleneck of the system. Try to imagine that the central server needs to handle millions of messages / transactions in seconds. It is not opportunistic to develop a scalable system in such a central way. Moreover, it is the single point of failure in the system. One denial of service attack may easily take the system down. Finally, the central server may be in one partition by itself so other nodes

may not be able to communicate the server and the whole system breaks down. It is obvious that we need a distributed system to handle partitions.

A typical distributed transaction usually involves multiple participants that are located in different sites. In the ad-hoc partitions environment we described above, how could a transaction continue in different partitions? We may store the transaction state information such as intermediate results, and messages such as functions expected to be invoked in the target peer sites.

In the passive way, we store this information in the connected participating peer sites. When the peer site (PS) migrates into the partition with the targeted peer sites (TPS), PS will forward the state information and messages to the target peer sites. TPS checks the messages, runs the expected portion of the transaction, compares the results with state information, synchronizes the state information, and updates the version of the state information.

In a proactive way, we store this information in the messenger sites that provide the special delivery service that is similar as transportation delivery services provided by Fedex and UPS. These messengers follow (roam) certain routes, which may pass through different partitions. The messenger stores the transaction state information and associated messages in the buffer, carries these information, and forwards these information to the destination when they meet the targeted sites.

A few issues still remain in the messenger approach: How frequent should the messengers run? How many messengers should the system run? How many "highways" should we build? How much should the messengers charge for the delivery of the

transaction information? How we could maximize the efficiency of the messenger? How to guarantee the different levels of quality of service (terminology similar as same day delivery, overnight delivery, second-day delivery, express saving delivery and ground shipping)?

8.1.2 Agent approach

In this approach, special agents with state information and messages are sent to peer sites and run the code and bring the new state information.

8.1.3 Data reservation approach

The site reserves part of the data during a certain time period. For example, travel agents may reserve some seats from the airline flight database. When network partitions happen, reserve transactions could still be completed. Data reservation approach may not work if participating sites endure a long duration. So the reservation period may expire and the other sites may use and update the previous reserved data. This gives rise to inconsistency problems among sites. For example, it may cause overbooking in the above travel agents.

8.2 Other Future Work

The future work in the SyD middleware mainly focus on integrating web services, supporting broader mobile devices and data stores, supporting Quality of Service at the different levels of applications.

The future work in quality aware transaction processing framework is to support rapid development of quality aware transaction services, provide the QoS specification

template for different domains, evaluate the techniques in real emergency response applications, and conduct additional evaluations for the probabilistic concurrency control mechanism and the group based transaction commit protocol.

The future work in PeerDS is to integrate with SyD middleware, integrate with quality aware transaction processing framework, deal with replication in P2P network, deal with fuzzy matching of queries in DHT based P2P network, and deal with node failures. In addition, we want to further evaluate our load balancing schemes.

The future work in filter indexing is to study complex filters such as statistics filters and investigate different update scheduling schemes.

CHAPTER 9 CONCLUSIONS

Developing collaborative applications over mobile and heterogeneous platforms is a very challenging task. Traditional transaction processing techniques cannot fully model and handle the dynamism and scalability associated with such applications. In this dissertation, we identify four research thrusts towards this goal.

First, we propose a multi-state transaction model for transaction processing over a collection of heterogeneous, possibly mobile data stores. Based on this model, we develop a quality aware transaction processing framework to incorporate quality of service with transaction processing. In order to support quality aware transactions, we redefine and adapt the traditional concepts such as atomicity, consistency, isolation and durability to suit the proposed environment so that transactions can be made flexible and adaptable. We also develop a quality specification language to associate quality of service with transaction properties. On the whole, we support disconnection-tolerant, and partition-tolerant transaction processing without assuming that a central server is always available.

Second, we develop a probabilistic concurrency control mechanism and group based transaction commit protocol in which we utilize the probability and feedback to adaptively find the balance between cascading abort and the long blocking in concurrent transaction control. This reduces blockings in transactions and improves the transaction commit ratio.

Third, we develop a scalable, efficient and highly available directory service called PeerDS to support the above framework as many mobile devices could provide services and participate in a distributed transaction. We address the scalability and dynamism of the directory service from two aspects: peer-to-peer and push-pull hybrid interfaces. We also address peer heterogeneity and load balancing in the peer-to-peer system. We optimize the routing algorithm in a virtualized peer-to-peer overlay network and develop a generalized Top-K server selection algorithm for load balancing, which could be optimized on different factors such as proximity and cost. From push-pull hybrid interfaces aspect, we propose to add a push interface, in addition to the conventional pull interface, to reduce the overhead of directory servers caused by frequent queries of directory clients.

Last, we study and evaluate different filter indexing schemes to improve the scalability and update scheduling of large subscription-based systems (e.g., the push-pull hybrid interfaces in PeerDS) as the filtering process of the updates might become the bottleneck.

These techniques extend the capabilities of key components of our System on Mobile Devices (SyD) middleware, which enables collaborative distributed applications over heterogeneous mobile handheld device and data stores.

The work described in this dissertation has been implemented and published in the following conferences: the 10th International Conference on Database Systems for Advanced Applications (DASFAA 2005), the 11th International Conference on Database Systems for Advanced Applications (DASFAA 2006), the 5th International Middleware

Conference (Middleware 2004), Symposium of Database Management in Wireless Network Environments in the 58th IEEE Vehicular Technology Conference (VTC 2003 Fall), and Workshop on Mobile Distributed Computing in The 23rd International Conference on Distributed Computing Systems (ICDCS 2003). Additional materials could also be found in Technical Reports in College of Computing, Georgia Institute of Technology.

REFERENCES

[1] D. Agrawal, J. L. Bruno, A. E. Abbadi and V. Krishnaswamy, "Relative Serilizability: An Approach for Relaxing the Atomicity of Transactions", ACM Symposium on Principles of Database Systems, 1994.

[2] W. Balke, W. Nejdl, W. Siberski, U. Thaden. " Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks", Proceedings of ICDE 2005, Tokyo, Japan.

[3] D. Barbar'a. "Certification Reports: Supporting Transactions in Wireless Systems." In the IEEE International Conference on Distributed Computing Systems (ICDCS), 1997

[4] P.A. Bernstein, V. Hadzilacos, and N. Goodman, **Concurrency Control and Recovery in Database Systems**, Addison-Wesley, Massachusetts, 1987.

[5] N.T. Bhatti and R.D. Schlichting. "Configurable Communication Protocols for Mobile Computing", in the 4th International Symposium on Autonomous Decentralized Systems, Tokyo, March 1999.

[6] M. Carey, S. Krishnamurthi and M. Livny, "Load Control for Locking: The 'Half-and-half' Approach", in the 9th Symp. On Principles of Database Systems (PODS), April 1990.

[7] J. Chen, D. J. Dewitt, F. Tian and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases". In ACM SIGMOD, 2000.

[8] P. K. Chrysanthis, "Transaction Processing in a Mobile Computing Environment", in IEEE workshop on Advances in Parallel and Distributed Systems, October 1993.

[9] CNN. http://money.cnn.com/2003/08/15/technology/landlines/ August 16, 2003

[10] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. "Wide-area cooperative storage with CFS." Proceedings of the 18th ACM Symp. On Operating Systems Principles (SOSP 01), Oct. 2001.

[11] Dirckze, R. and L. Gruenwald, "A Pre-Serialization Transaction Management Technique for Mobile Multidatabases", ACM Mobile Networks and Applications, Volume 5, Number 4, December 2000, pp. 311-321.

[12] Dirckze, R. and L. Gruenwald, "A Toggle Transaction Management Technique for Mobile Multidatabases", ACM Conference on Information and Knowledge Management (CIKM), November 1998, pp. 371-377.

[13] Eddie Y.M. Chan, Victor C.S. Lee, and Kwok-Wa Lam , "Using Separate Processing for Read-Only Transactions in Mobile Environment", LNCS 2574, pp 106-121,2003.

[14] W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith, S. Izadi, "Recombinant computing and Speakeasy Approach," in MobiCom 2002, Atlanta, GA. USA.

[15] M.H. Eich and A. Helal, "A Mobile Transction Model that Captures Both Data and Movement Behavior", ACM-Baltzer Journal on Special Topics on Mobile Networks and App, 1997.

[16] G. Eisenhauer, F. Bustamente and K. Schwan, "A Middleware Toolkit for Client-Initiated Service Specialization", in the PODC Middleware Symposium, 2000.

[17] G. Eisenhauer. "The ECho Event Delivery System", Technical Report GIT-CC-99-08, College of Computing, Georgia Institute of Technology.

[18] A. K. Elmagarmid(ed), Database Transaction Models for Advanced Applications ,Morgan-Kaufmann, 1992.

[19] R. Elmasri and S.B. Navathe, **Fundamentals of Database Systems**, Addison Wesley, Ed. 4, 2004.

[20] F. Fabret etl., "Flistering Algorithms and Implemention for Very Fast Publish/Subscribe Systems", in ACM SIGMOD, 2001.

[21] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets", in ACM SIGCOMM, 2003.

[22] A. Farrag and M. Ozsu, "Using Semantic Knowledge of Transactions to Increase Concurrency", ACM Transactions on Database Systems, 14(4), Dec 1989.

[23] S. Frolund and J. Koistinen, "Quality-of-Service Specification in Distributed Object Systems", Distributed System Engineering Journal, 5(4), Dec 1998.

[24] L. Galanis, Y. Wang, Shawn R. Jeffery, David J. DeWitt. "Locating Data Sources in Large Distributed Systems." VLDB 2003.

[25] H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database", ACM Trans. Database System 8, 2 (June 1983), 186-213.

[26] H. Garcia-Molina and K. Salem "Sagas", in ACM SIGMOD 1987.

[27] B. Gedik and L. Liu. "PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Infrmation Monitoring System". ICDCS 2003.

[28] J. N. Gray, "The transaction concept: virtues and limitations", in VLDB 1981

[29] http://gnutella.wego.com Oct 2003

[30] X. Gu and K. Nahrstedt, "Visual Quality of Service Programming for Distributed Heterogeneous Systems ", Technical Report UIUCDCS-R-2000-2190, Univ. of Illinois at Urbana-Champaign, Nov. 2000

[31] R. Gupta, J. Haritsa, K. Ramamritham, "Revisiting commit processing in distributed database systems", in ACM SIGMOD 1997

[32] M.A. Hiltunen, R.D. Schlichting and G. Wong, "Implementing Integrated Fine-Grain Customizable QoS using Cactus", in the 29th Annual International Symposium on Fault-Tolerant Computing, Madison, WI, June 1999.

[33] T. Imielinksi and B. R. Badrinath, "Wireless Mobile Computing: Challenges in Data Management", Communications of the ACM, October 1994, 37(10), pp. 18-28.

[34] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web". In ACM Symposium on Theory of Computing Author Index, pages 654–663, May 1997.

[35] V. Kumar, N. Prabhu, M. H. Dunham, A. Y. Seydim, "TCOT-A Timeout-Based Mobile Transaction Commitment Protocol", IEEE transactions on Computers , VOL 51, NO.10 OCTOMBER 2002

[36] G. Kortuem, J. Schneider, D. Preuitt, T. Thompson, S. Fickas and Z. Segall. "When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad-hoc Networks," First International Conference on Peer-to-Peer Computing (P2P 2001), Linköpings, Sweden, August, 2001, pp. 75-91.

[37] G. Kortuem. "Proem: A Peer-to-Peer Computing Platform for Mobile Ad-hoc Networks," Advanced Topic Workshop Middleware for Mobile Computing, Heidelberg, Germany, November, 2001.

[38] Y. Krishnamurthy, V. Kachroo, D.A. Karr, C. Rodrigues, J.P. Loyall, R.E. Schantz, D.C. Schmidt. "Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications." in ACM

SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.

[39] V. Kumar, B. F. Cooper, S. B. Navathe, "Predictive Filtering: A Learning-Based Approach to Data Stream Filtering", in Workshop of Data Management for Sensor Networks in conjunction with VLDB 2004.

[40] B. Lampson and H. Sturgis, "Crash recovery in a distributed data storage system," Technical Report, Xerox Palo Alto Research Center, 1976.

[41] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations", IEEE Journal of Selected Areas in Communication, Special Issue on Service Enabling Platforms, vol. 17, num. 9, pp. 1632-1650, September, 1999.

[42] J. Loyall, D. Bakken etc., "QoS Aspect Languages and Their Runtime Integration." Proc. of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), May 1998.

[43] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems" Proceeding of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98), Kyoto, Japan, 20-22 April 1998.

[44] Q. Lu and M. Satyanarayanan. Isolation Only Transactions for Mobile Computing. Operating Systems Review, pages 81--87, April 1994

[45] N. Lynch, "Multilevel Atomicity – a New Correctness Criterion for Database Concurrency Control", ACM Transactions on Database Systems, 8(4), Dec 1983.

[46] S. Madden, M. Shah, J. M. Hellerstein and V. Raman, "Continuously Adaptive Continuous Queries over Streams". In ACM SIGMOD, 2002.

[47] K. Nahrstedt, D. Xu, D. Wichadakul and B. Li, "QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments", IEEE Communications Magazine, 2001.

[48] New York City Emergency Response Task Force, http://www.nyc.gov/html/om/pdf/em_task_force_final_10_28_03.pdf, October 28, 2003

[49] J. Newmarch, "A Programmer's Guide to Jini Technology", A Press, 2000.

[50] C. Olston, J. Jiang and J. Widom, "Adaptive Filters for Continuous Queries over Distributed Data Streams". In ACM SIGMOD, 2003.

[51] T. Phan, L. Huang, C. Dulan , "Integrating Mobile Wireless Devices Into the Computational Grid," in Proc. of MobiCom 2002, Atlanta, GA. USA, September 23-28, pp. 271 – 278.

[52] E. Pitoura and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. In Proceedings of the 15th IEEE International Conference on Distributed Computing Systems, pages 404--413, May 1995.

[53] E. Pitoura and B. Bhargava. A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases. ACM SIGMOD Record, 24(3): 44--49, September 1995

[54] E. Pitoura and B. Bhargava, "Building Information Systems for Mobile Environments", in the 3rd Intenational Conference on Information and Knowledge Management (CIKM), 1994

[55] S. K. Prasad, V. Madisetti, R. Sunderraman, et al. "System on Mobile Devices (SyD): Kernel Design and Implementation", in First International Conference on Mobile Systems, Applications, and Services (MobiSys), Poster and Demo Presentation, May 5-8, 2003, San Francisco.

[56] S. K. Prasad, A. G. Bourgeois, E. Dogdu, et al. "Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile Data Stores Using SyD Link Technology", in Mobile Wireless Network Workshop held in conjunction with The 23rd International Conference on Distributed Computing Systems (ICDCS) , May 19-22, Providence, Rhode Island.

[57] S. K. Prasad, A. G. Bourgeois, E. Dogdu, et al. "Implementation of a Calendar Application Based on SyD Coordination Links", in the Third International Workshop on Internet Computing and E-Commerce in conjunction with the 17th Annual International Parallel & Distributed Processing Symposium (IPDPS), April 2003, Nice, France.

[58] S. K. Prasad, E. Dogdu, R. Sunderraman, et al., "Design and Implementation of a listener module for handheld mobile devices", in the 41st Annual ACM Southeast Conf., Savannah, Georgia, March, 2003.

[59] S. K.Prasad, M. Weeks, Y. Zhang, et al., "Mobile Fleet Application Using SOAP and System on Devices (SyD) Middleware Technologies," Communications, Internet and Information Technology (CIIT 2002), Virgin Islands, USA, November 18-20, 2002, pages 426-431.

[60] S. K. Prasad, V. Madisetti, S.B. Navathe, R. Sunderraman, E. Dogdu, A. Bourgeois, M. Weeks, B. Liu, J. Balasooriya, A. Hariharan, W. Xie, P. Madiraju, S. Malladi, R. Sivakumar, A. Zelikovsky, Y. Zhang, Y. Pan, and S. Belkasim, "SyD: A Middleware

Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores", in ACM/IFIP/USENIX 5th International Middleware Conference (MW-04), Toronto, Oct. 2004.

[61] N. Preguiça, J. L. Martins, M. Cunha and C. Baquero, et al. "Mobile Transaction Management in Mobisnap". In the 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems – Special Sessions on Mobile Database Technology (LNCS 1884), September 2000.

[62] N. Preguiça, J. L. Martins, M. Cunha and C. Baquero, "Reservations for Conflict Avoidance in a Mobile Database System". In the 1st International Conf. On Mobile Systems, Applications, and Services, May 5-8, 2003, San Francisco, CA.

[63] A. Rao, K. Lakshminarayanan, R. K. Sonesh Surana, and I. Stoica. "Load balancing in structured p2p systems." In the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Feb. 2003.

[64] S. Ratnasamy, P. Francis, M. Handley, et al., "A Scalable Content-Addressable Network." In SIGCOMM 2001.

[65] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November 2001.

[66] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing ", Fifteenth ACM Symposium on Principles of Distributed Computing May 1996, Philadelphia, PA.

[67] A.H. Skarra and S. B. Zdonik, "Concurrency control and object-oriented databases", in Object-Oriented Concepts, Databases and Applications, ACM Press, pp 395-421

[68] D. Skeen, "Nonblocking commit protocols," in ACM SIGMOD 1981, pp.133-142

[69] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications." SIGCOMM 2001.

[70] System on Devices (SyD): A Model with Coordination Links and a Calendar Application, Utility Patent Filed,2002

[71] W. Tang, L. Liu and C. Pu, "Trigger Grouping: A Scalable Approach to large Scale Information Monitoring", In NCA 2003.

[72] R. Vanegas, J.A. Zinky, J.P. Loyall, et al., “QuO's Runtime Support for Quality of Service in Distributed Objects”. In the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, England, September 1998.

[73] G. Walborn and P. K. Chrysanthis. “Supporting Semantics Based Transaction Processing in Mobile Database Applications, ”. In the 14th Symposium on Reliable Distributed Systems, September 1995.

[74] G. Walborn and P. K. Chrysanthis. “PROMOTION: Support for Mobile Database Access”, Personal Technologies Journal, 1(3), September 1997.

[75] B. Wilson, “**JXTA**”, New Riders Publishing, 2002.

[76] W. Xie , S. B. Navathe, S. K. Prasad., "Supporting QoS-Aware Transaction in the Middleware for a System of Mobile Devices (SyD)," in the 1st International Workshop on Mobile Distributed Computing in The 23rd International Conference on Distributed Computing Systems (ICDCS), May 19-22, 2003 Providence, Rhode Island.

[77] W. Xie , S. B. Navathe, "Transaction Adaptation in System on Mobile Devices (SyD): Techniques and Languages," in Symposium of Database Management in Wireless Network Environments in the 58th IEEE Vehicular Technology Conference (VTC03 Fall), Oct 7-10, 2003, Orlando, Florida.

[78] W. Xie, S. B. Navathe and S. K. Prasad, “Filter Indexing: a Scalable Solution to Large Subscription Based Systems”, in the 10th International Conference on Database Systems for Advanced Applications (DASFAA), Apr 2005.

[79] W. Xie, S. B. Navathe and S. K. Prasad, “PeerDS: a scalable directory service”, in submission.

[80] W. Xie, S. B. Navathe and S. K. Prasad, “Optimizing Peer Virtualization and Load Balancing”, To appear in the 11th International Conference on Database Systems for Advanced Applications (DASFAA), Apr 2006.

[81] W. Xie, S.B. Navathe and S. K. Prasad, “Supporting distributed transactions over dynamically partitioned networks”, in preparation.

[82] T. W. Yan and H. Garcia-Molina, “Index Structures for Selective Dissemination of Information Under the Boolean Model”, in ACM Transactions on Database Systems, VOl 19 N0.2 June 1994.

[83] B. Yang and H. Garcia-Molina. “Designing a super-peer network.” In Proc. ICDE, March 2003.

[84] L. H. Yeo and A. Zaslavsky. “Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment.” In the 14th International Conference on Distributed computing Systems (ICDCS), Poznan, Poland, June 1994.

[85] B. Yoshimi, N. Sukaviriya, H. Derby, B. Carmeli, B. Bolam, J. Elliott, J. Morgan . "Lessons Learned in Deploying a Wireless, Intranet Application on Mobile Devices", Fourth IEEE Workshop on Mobile Computing Systems and Applications June 20 - 21, 2002.

[86] B. Y. Zhao, L. Huang, J. Stribling, et al., “Tapestry: A Resilient Global-scale Overlay for Service Deployment”, IEEE Journal on Selected Areas in Communications.

[87] Y. Zhu and Y. Hu, “Towards Efficient Load Balancing in Structured P2P Systems”, in IPDPS 2004.

Vita

Wanxia Xie was born in Hu-Nan, China. He attended Beijing Institute of Technology, majoring in Flight Vehicle Engineering with a minor in computer science. He then went to Department of Computer Science, Peking University for graduate study. He obtained a Master of Science degree in computer science. He joined the Ph.D. program in College of Computing, Georgia Institute of Technology, Atlanta, Georgia in 1999. Along the way in the Ph.D. program, he received a Master of Science degree in Computer Science in 2001.