

Enabling Scalable Information Sharing for Distributed Applications Through Dynamic Replication

A Thesis
Presented to
The Academic Faculty

by

Tianying Chang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005

Enabling Scalable Information Sharing for Distributed Applications Through Dynamic Replication

Approved by:

Professor Mustaque Ahamad
College of Computing, Adviser

Professor Jim (Jun) Xu
College of Computing

Professor Ling Liu
College of Computing

Dr Zhen Liu
IBM T.J Watson

Professor Kishore Ramachandran
College of Computing

Date Approved: Nov 28th, 2005

To my parents, for everything.

ACKNOWLEDGEMENTS

I would first like to thank my dissertation committee members, Ling Liu, Kishore Ramachandran, Jim Xu and Zhen Liu, for their time and suggestions. I greatly appreciate their insight and comments that have influenced and improved my research.

I am so fortunate to be a part of the System research group, from which I have received not only inspiration and technical support but also encouragement and friendship during the years. Zhiyuan Zhan, Yuan Chen, Seung Jun, and Sameer Adhikari provided me with very useful feedback on my research on many occasions. Qi He, Weiling Zhu, Yarong Tang and Donghua Xu offered a lot of help in my everyday life. All the members of the System group have played an important role in making my years in the CoC building pleasant and unforgettable.

I also want to thank Chris Codella and Zhen Liu for providing me the opportunity to work at IBM T.J. Watson Research Center where I developed part of the results forming this thesis. My fiance Jinliang Fan is a big reason why I am writing this section of my dissertation today. In both research and life, his support followed me throughout my Ph.D. years and he brought much more fun to this long journey.

Finally, and most importantly, I want to express my sincere gratitude for my advisor, Mustaque Ahamad, for his guidance, support, and patience throughout my Ph.D. years. I feel extremely lucky to have Mustaque as my advisor, who is always dedicated to the success of his students. He provided me with the flexibility to choose projects and steered me in the right direction through valuable feedback and constructive criticism. He taught me the value of perseverance as well as the value of vision, the value of action as well as the value of direction. I have certainly grown throughout this study due to his words of wisdom, encouragement and faith in me.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
SUMMARY	xi
I INTRODUCTION	1
1.1 Motivation	1
1.1.1 Peer-to-Peer System	3
1.1.2 Distributed Object Middleware	3
1.1.3 Building Scalable Services With Non-Dedicated Resources	5
1.2 Research Challenges	7
1.2.1 Efficient Replication Schemes	7
1.2.2 Middleware Support	8
1.2.3 Scalable Update Dissemination	8
1.3 Dissertation Overview	9
1.3.1 Thesis	9
1.3.2 Thesis Contribution	9
1.3.3 Thesis Outline	11
II BACKGROUND AND RELATED WORK	13
2.1 Data and Service Replication	13
2.1.1 Overview	13
2.1.2 Replication Schemes	15
2.1.3 Replication in Distributed Object Systems	17
2.2 Peer-to-Peer Systems	19
2.2.1 Peer-to-Peer Technologies	19
2.2.2 Peer-to-Peer Middleware	19
2.3 Advanced Network Technologies for Group Communication	20
2.3.1 Overlay Networks	20

2.3.2	IP Multicast and Application Layer Multicast	21
2.3.3	Preference-Aware Communication	22
2.4	Summary	24
III	AURA: AUTONOMOUS REPLICATION ALGORITHM	25
3.1	Introduction	25
3.2	Peer-to-Peer Based Object Replication	27
3.2.1	Performance	28
3.2.2	Controlled Number of Replicas	28
3.2.3	Autonomous Adaptability	29
3.2.4	Fairness	29
3.3	AURA: An Autonomous Replication Algorithm	30
3.4	Performance Evaluation	38
3.4.1	Simulation Setup	38
3.4.2	Stability of Replica Fraction	39
3.4.3	Fairness of Resource Contribution	40
3.4.4	Average Network Latency Between Replicas and Client Peer Nodes	40
3.4.5	Experiments with PlanetLab	42
3.5	Summary	44
IV	UPDATE DISSEMINATION USING LIMITED NUMBER OF MULTICAST CHANNELS	45
4.1	Introduction	45
4.2	Group Arrangement Problem	47
4.2.1	Example: Massive Multiplayer Game	47
4.2.2	Layered Preferences and Adaptive Grouping	47
4.2.3	Examples of Group Arrangement	48
4.2.4	Problem Definition	49
4.3	A Heuristic Searching Algorithm	51
4.3.1	General Idea	51
4.3.2	Two Phases of A Single Adjustment Step: Splitting and Merging	53
4.3.3	Computation Complexity	55
4.4	Evaluation	55

4.4.1	Generating Preference matrix	55
4.4.2	Comparison With Cell-based Algorithm	57
4.4.3	Joins and Leaves During Regrouping	58
4.4.4	Connection Similarity Vs. Receiver(source) Similarity	59
4.5	Summary	60
V	UPDATE DISSEMINATION WITH OVERLAY NETWORK	63
5.1	Introduction	63
5.2	Problem Formalization	66
5.3	Constructing Preference-Aware Overlay Topologies	68
5.3.1	Multi-Attribute Clustering of Nodes	69
5.3.2	Constructing Overlay Topology Based on Clustering of Nodes	71
5.4	Performance Evaluation	74
5.4.1	Experiment Setup	74
5.4.2	Impact of Clustering on Latency and Waste	76
5.4.3	Impact of Overlay Network Scale on Latency and Waste	81
5.5	Summary	82
VI	GT-P2PRMI	84
6.1	Introduction	84
6.2	GT-P2PRMI: An Extension of Java RMI Framework	85
6.2.1	User Interface of GT-P2PRMI	86
6.2.2	Implementation of GT-P2PRMI	87
6.3	Performance Studies	90
6.3.1	Experimental Evaluation in Small Scale Environment	90
6.3.2	Simulation of Large Scale Internet Environment	91
6.4	Summary	93
VII	CONCLUSIONS AND FUTURE WORK	94
7.1	Contributions	94
7.2	Future Work	96
	REFERENCES	98
	VITA	107

LIST OF TABLES

1	Parameters Used in Algorithm	34
2	Requirement Conditions	49
3	Case 1	49
4	Case 2	49
5	Case 3	49
6	Different Latency and Waste for Different Overlay Topologies	65

LIST OF FIGURES

1	Dedicated servers are used to replicate a service.	5
2	Different peer nodes replicate the service at different time	6
3	State Transition Diagram	33
4	Variation of the number of replicas in the system when sleep ratio ϕ_i of each node u_i is fixed	39
5	Variation of the number of replicas in the system when sleep ratio ϕ_i of each node u_i is adjustable	40
6	Total amount of time that each node contributed to replicate the service . .	41
7	Average latency between peer nodes and their nearest replicas when different fraction of replicas are created	41
8	Variation of the number replicas in the system at the same time	43
9	CDF of the average RMI call response time of all the peer nodes	43
10	Example of Preference and Subscription Matrices	50
11	Split/Merge Greedy Heuristic Algorithm	52
12	Connections Partition	54
13	Diverse Receivers and Sources	56
14	Comparison with Cell-based Grouping	57
15	Comparison with Cell-based Grouping	58
16	Comparison with Cell-based Grouping	59
17	Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR) . .	60
18	Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR) . .	61
19	Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR) . .	61
20	Example of Different Overlay Topologies and Different Data Path	65
21	Generation of Preference Type I and II	75
22	CDF of Latency Penalty When $\omega = 0$	76
23	CDF of Waste for Preferences Type I When $\omega = 1$	77
24	CDF of Waste for Preferences Type II When $\omega = 1$	77
25	CDF of Latency When $\omega = 0.5$	78
26	CDF of Waste When $\omega = 0.5$	78
27	Latency Affected by Clustering Parameter ω	79

28	Waste Affected by Clustering Parameter ω	80
29	Combination of Latency and Waste When $\lambda = 0.5$	80
30	Latency Versus Size of Network	81
31	Waste Versus Size of Network	82
32	Illustration of Remote Invocation in GT-P2PRMI	89
33	Invocation Latency Affected by Replication Probability	92
34	Invocation Latency Affected by Lookup Frequency	92

SUMMARY

As broadband connections to the Internet become more common, new information sharing applications that provide rich services to distributed users will emerge. Furthermore, as computing devices become pervasive and better connected, the scalability requirements for Internet-based services are also increasing. Distributed object middleware has been widely used to develop such applications since it makes it easier to develop distributed applications for heterogeneous computing and communication systems. As the application's scale increases, however, the client/server architecture limits the performance due to the bottleneck at the centralized servers. The recent development in peer-to-peer technologies creates a new opportunity for addressing scalability and performance problems for services that are used by many nodes. In a peer-to-peer system, peer nodes can contribute a fraction of their resources to the system, enabling more flexible and extended sharing between the entities in the system. When peer nodes are required to contribute their resources by replicating a service for self and others, however, several new challenges arise.

Our thesis is that non-dedicated resources in a distributed system can be utilized to replicate shared objects dynamically so that the quality and scalability of a distributed service can be achieved with lower cost by replicating the objects at right places and by disseminating the updates to those objects efficiently and quickly. The following are the contributions of our work that has been done to validate the thesis.

- A new fair and self-managing service replication algorithm that allows distributed non-dedicated resources to be used to improve service performance with lower cost.
- A preference-aware multicast grouping algorithm that is used to disseminate updates to the shared objects among a large set of heterogeneous peer nodes to keep a consistent view for all peer nodes.

- An preference-aware overlay construction algorithm that aims at reducing both network latency and total network traffic when delivering data through the built overlay network.
- An implementation of a distributed object framework, GT-RMI, that allows peer nodes to invoke dynamically replicated objects transparently. The framework can be configured for a particular peer node through a policy file.

Our extensive evaluation of the techniques proposed in this dissertation demonstrates that our approach is viable for building scalable distributed services.

CHAPTER I

INTRODUCTION

1.1 Motivation

Over the past decade, the Internet has become a ubiquitous medium of communication, providing connectivity between computing devices that are both geographically and administratively distributed. The increasing penetration of the Internet into offices and households allows millions of users to access a variety of Internet-based services and distributed applications, and has led to the growth of these services and applications. Distributed interactive applications like massive multiplayer games, virtual reality video conference, virtual shopping malls, battle field simulations, and collaborative work environments connect remote producers and consumers of information together in real time over the Internet. The heterogeneity of the Internet along with real-time requirements and large number of users complicate the development and deployment of such applications.

An example of distributed interactive applications that require scalable information sharing is massive multiplayer game (MMG). MMGs involve a large number of players, and state updates must be disseminated in a timely manner in order for the game to be playable. As MMGs become more sophisticated, they demand more network resources or demand using existing network resources more efficiently. In a MMG, players interact with each other in the same virtual world. This virtual world consists of many game entities: the avatars of players, non-player monsters, and props. At any given time, each player is only interested in a relatively small subset of all the entities, and can be interested in each game entity at varying levels of detail. Each game entity has state associated with it, for example, location, posture, and physical condition. As the state of a game entity changes, state updates must be disseminated to players who would notice the change.

From the application development perspective, a good programming interface for distributed applications is needed so that it can reduce the cost of application development.

From the system architecture perspective, the resulting system should be scalable to the expected number of users who will interact with each other. From the operational perspective, the resulting system should adapt to the dynamically changing users, in terms of both their population and their locations.

Distributed object platforms, e.g., CORBA, DCOM and RMI, provide middleware support for distributed applications and has become popular since it provides a convenient and familiar programming interface by replicating servers that provide key services. For example, when a game application is implemented over a distributed object platform, game entities could be designed as a set of objects, e.g. locations, outlines, colors and textures, which define different details about the entity. However, distributed object platforms do not meet the real-time requirements of distributed interactive applications over the Internet. The state of a needed object is transferred “on demand” to the remote user via an RPC call. The communication induced by RPC not only results in high latency but the scalability of the server could be limited due to its interaction with many clients. Scalability and performance can be improved by increasing the number of dedicated servers when needed. For example, a game object can be implemented at a game server and replicated at other replication servers. Updates to an object are disseminated to the servers that are replicating this object, and different consistency schemes among the object and its replicas can be used to provide different level of motion smoothness in gaming.

However, the cost of deploying large number of servers increases as the demands of participating users increase. It is relatively harder to adapt the number of dedicated servers to the dynamically changing users, in terms of both the population and locations. Peer-to-peer systems have offered the potential to provide resources that can grow with the dynamically changing users with low cost. However, a fully distributed peer-to-peer system introduces other challenges. We will discuss briefly the benefits and drawbacks of such systems below and then talk about our idea to combine the benefit dedicated servers and peer-to-peer systems.

1.1.1 Peer-to-Peer System

An important consequence of the ubiquitous Internet is the opportunity to share with total strangers not only communication infrastructure but also other resources, such as computers and data. The classic example is Seti@home [4]: a project that benefits from resources available on computers all around the world. Other examples are the peer-to-peer music-sharing networks such as Napster [3], Kazaa [2], or Gnutella [6]: a file provided by a peer in Atlanta is downloaded by a user in Seattle, this operation being made possible by many other peers from, for instance, Germany, France and Canada. The participants in peer-to-peer network provide unprecedented computing power and bandwidth. Those resources are distributed and scale with the size of the participating users. If those resources can be collected and utilized properly, it is possible to improve the quality of services with low cost.

Sharing resources over the Internet, however, raises many technical problems, amplified by the potential scale of the resource pool, the heterogeneity of platforms, the diversity in user behavior, and the inherent lack of reliability. The distributed resources have to be published and discovered; the resources cannot be assumed as reliable as the dedicated resources; the free riders are not desired; and so on. On one hand, we prefer to take advantage of the large amount of potential resources available at peer nodes. On the other hand, we do not want users of the service to be interrupted frequently. We also do not want the application developers to be distracted by the underlying peer-to-peer network details.

1.1.2 Distributed Object Middleware

Traditionally, services are hosted by centralized servers. Particularly, for the sake of programming and development efficiency, many of these services are implemented over distributed object platforms, e.g. middleware systems like CORBA, RMI, DCOM and SOAP [22, 78, 104, 109]. In these systems, services are implemented as objects. The centralized servers are responsible for maintaining these objects and processing remote requests from distributed users. Users submit their requests by invoking remote procedure calls (RPCs) on remote objects and wait for responses. The servers process the requests by invoking

called methods of the objects and send the results back to the users. In this architecture, all requests from the users are sent to the central servers.

A middleware platform for distributed objects can provide great convenience for client/server based applications across heterogeneous users. It facilitates the development of distributed applications by allowing an application developer to design and build applications using (standard) object-oriented techniques, without having to deal with the complexity due to the distribution of objects and components. However, there are two potential drawbacks in such an architecture. First, when a large number of users send requests to the central servers at the same time, the central servers could be overloaded, therefore, the quality of the service degrades. Second, this architecture lacks locality. Requests from users that are located faraway from the central servers could incur long service response time due to the large network latency.

To reduce the response time of distributed services hosted by central servers, researchers have proposed the use of object replication and caching to improve the performance of distributed object systems [60, 63, 40]. In this approach, objects maintained on the servers are fully [63] or partially [40] cached or replicated at multiple locations. When it is necessary or preferable, remote calls are directed to locally cached copies instead of the objects at far away servers. The response time of remote call thus is significantly reduced and the overall system performance is improved. To maintain the consistency among cached or replica copies of objects, various consistency protocols are used with different consistency maintenance overhead and performance gain. Although replication and caching can reduce the impact of communication latency on the response time of the remote calls, the fact that object states are cached or replicated by all clients implies an added cost for maintaining consistency. This often forces the system to adopt a weaker consistency model so that the consistency maintenance overhead is at an acceptable level, especially in systems with a relative high ratio of write/read object operations [63].

One technique that can improve the scalability and reduce the service response time for applications without caching and replicating at every user is to deploy a set of dedicated servers at distributed locations to host the service. When enough dedicated servers are

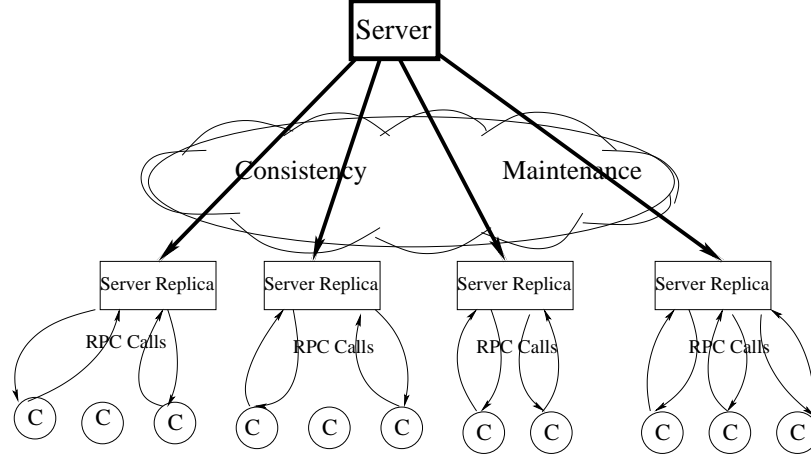


Figure 1: Dedicated servers are used to replicate a service.

deployed, there should be enough service capacity to meet the demands from all the users. Consistency needs to be maintained among the set of dedicated servers. As shown in Figure 1, a set of dedicated servers are deployed at geographically distributed locations and they replicate the service from the central server. Each client node sends its requests to the closest server replica. The set of server replicas need to maintain consistency. The provisioning of dedicated replicated servers, however, is not trivial — efficient utilization of resources is difficult to achieve especially when the loads are heterogeneously distributed and dynamically changing. First, the number of dedicated servers must be provisioned. There should be enough servers so that even the peak service load can be handled. However, provisioning too many servers causes resource waste when the service load is at non-peak levels. Second, the locations of these dedicated servers also need to be decided. A random or uniform placement of servers may not be able to fully utilize resources under heterogeneous distribution of peer node requests. Furthermore, when the request distribution changes dynamically, a static placement of servers has limited adaptability.

1.1.3 Building Scalable Services With Non-Dedicated Resources

To provide scalable distributed service while maintaining the ease of software development, we explore the idea of replicating distributed objects dynamically at participating peer nodes. Such peers differ from dedicated servers because they can choose not to provide the

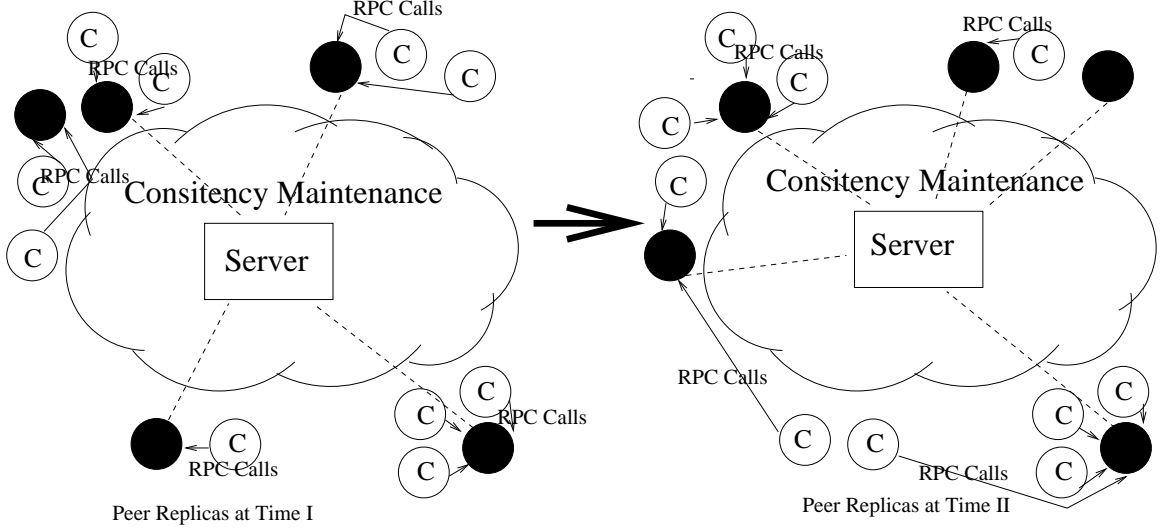


Figure 2: Different peer nodes replicate the service at different time

service when their total resources are limited.

In a peer-to-peer system, peer nodes can contribute a fraction of their resources to the system, enabling more flexible and extended sharing among the entities in the system. First, contributing peer nodes can provide additional resources (often free or little cost) as the number of such peers grows. Second, peers can geographically extend the scope of the service where it is needed. This allows for more flexible and dynamic placement of service instances.

As shown in Figure 2, a subset of peer nodes choose to contribute their resources to replicate the service from a server and each peer node sends its RPC call to an available service replica in its vicinity. The peer nodes that create service replicas locally need to maintain consistency for the replicated service among themselves. Different peer nodes could be replicating the service at different times. Therefore, each peer node could invoke the service from different peer nodes at different times. Compared with the architecture using central server without caching, this architecture has two advantages. First, load of the central server can be shared by distributed replicas. Second, replicas can be created where there are demands for the service therefore increasing the locality of the service. Compared with the architecture that requires each peer node to cache services locally, this architecture

has lower consistency maintenance overhead. Compared with the architecture using a set of dedicated server replicas, this architecture offers the advantage of dynamically tuning the number and location of replicated service instances with greater flexibility.

1.2 Research Challenges

The idea of dynamically utilizing resources that are available at distributed peer nodes can potentially improve scalability and performance of distributed services with lower cost. When the resources of peer nodes are collected to replicate a service for self and others, however, several new challenges arise. Those challenges involve different aspects, including replication algorithm, support from the underlying middleware system and consistency requirements.

1.2.1 Efficient Replication Schemes

There are several challenges for placing replicas in peer-to-peer environment. First, how should we decide the placement of server object replicas based on the distribution of service requests? The placement should be dynamically adjusted when the distribution of service requests changes. The higher the demand for a service in an area is, the more peers with service instances are needed to replicate the service. In this way, the workload from all the peer nodes can be evenly distributed among all the volunteer peers, instead of overloading some peers while other replicas are not receiving enough requests. Second, how to maintain the number of peer nodes that are concurrently replicating the server objects at a desired level, e.g., a certain percentage of total number of peer nodes? A degree of replication that is too high may cause excessive consistency maintenance cost. Too few replicas may not provide enough service capacity, therefore causing overload at the replicating nodes. Third, how to achieve fairness among peer nodes in terms of their contribution to the system? The fairness is a basic principle of resource sharing and has been a major consideration in many other peer-to-peer based systems. Since the volunteer peers are not dedicated service providers, they are not obligated to provide services for other peer nodes and fairness is important to ensure that each consumer of the service also replicates the service and shares it with others. Each node should contribute some of its resources for others and it is reasonable

that peer nodes that generate higher workload should also contribute more resources for others. Fourth, how can each peer node make decision based on limited knowledge it has autonomously? Due to the nature of peer to peer system, centralized decision should be avoided. Each peer node can communicate and cooperate with other peer nodes, but shouldn't be forced to replicate a service when it dose not want to do it.

1.2.2 Middleware Support

A middleware that can support dynamic replication using peer-to-peer network is needed and the middleware should address the following questions. First, unlike dedicated server nodes, peer nodes may not always be present and can even refuse to provide service. The middleware should provide seamless invocation support so that the peer nodes will not feel service disruption under these conditions. This implies a new replica should be found and invocation should be redirected if the requests are rejected or the peer node has left. Second, new service instances must be created on demand. Thus, support for dynamic downloading and instantiating of service objects needs to be added. After a replica is created and ready to provide service for others, its availability should be publicized, so that others can use the newly created service. Third, it should also be compatible with existing object systems and its interface should be kept the same as much as possible for the benefit of easy programming.

1.2.3 Scalable Update Dissemination

Updates to replicated objects made by any peer node should be sent to all the other replicating nodes so that all replicas can maintain a consistent state of the shared service. Maintaining consistency among a set of nodes becomes more expensive when the scale of the application becomes larger, network resources are limited and peer nodes' preferences are heterogeneous. Updates should be disseminated in a timely fashion and with minimal network bandwidth consumption.

1.3 *Dissertation Overview*

In this section, we clearly define our thesis and highlight the contributions presented in this dissertation.

1.3.1 Thesis

Our thesis is that non-dedicated resources in a distributed system can be utilized to replicate shared objects dynamically so that the scalability of a distributed service can be achieved by replicating the objects at right places and by efficiently disseminating updates to those shared objects.

1.3.2 Thesis Contribution

The following are the contributions of our work that has been done to validate the thesis.

- We have developed a new *fair and self-managing replication algorithm* that allows distributed non-dedicated resources to be used to improve service performance with lower cost. Our algorithm aims at utilizing distributed non-dedicated resources in a fair and efficient way and place service replicas at appropriate nodes without global knowledge for better scalability. It has the following features:
 - **Stable Replica Fraction** number of needed server replicas fluctuates with the number of peer nodes that actually use the service. Our algorithm maintains a stable replica fraction f in the system so that the total processing power available within the distributed peer nodes grows linearly as the number of peer nodes increases.
 - **Autonomous Decision Making** Each peer node decides if it will create a local service instance or delete it on its own. The decision is made without global knowledge of other replicas and does not need to get consent from other nodes. This meets the requirement of distributed scalable system by avoiding centralized decision making node.

- **Fair Resource Contribution** To enforce the fairness of contributed resources from all peer nodes, each node remembers the time it has replicated the service and then rests for a period of time that is proportional to its contributed time. This meets the requirement of peer-to-peer system where each peer node volunteers to serve others when it is convenient for it to do so.
- **Low Response Time** By introducing coordination mechanism among replicating nodes, the algorithm adaptively and probabilistically creates more replicas when and where they are needed. Replicas that fit best to current request distribution receive more requests than others and remain active for longer period.
- We developed multicast grouping algorithm and overlay construction algorithms for disseminating updates among a set of replicating nodes. Both algorithms aim at reducing the network bandwidth usage and minimize the latency of disseminating data among the set of interested receivers, especially users with heterogeneous interests.
 - **Multicast Grouping Algorithm** that is used to disseminate updates to the shared objects among a large set of heterogeneous peer nodes to keep consistent view for all peer nodes. Our algorithm groups nodes with similar interests into same group and multicasts all the required data to the group so that the unwanted data received by each node can be minimized.
 - **Overlay Construction Algorithm** that aims at reducing both network latency and total network traffic when delivering data through the built overlay network. Our multi-attribute clustering algorithm can be tuned to adapt to application user's desired tradeoff between network latency and traffic (which implies the amount of money charged).
- We implement a distributed object framework, GT-RMI, that allows peer nodes to invoke dynamically replicated objects transparently. The user only needs to specify the desired service, the requests will be directed to selected replicas. If the invocation fails, it will be redirected to other available service provider. Any peer node can

replicate the object and advertise it to the public. The framework can be configured for a particular peer node through a policy file and has the following characters:

- **On-Demand Service Replication:** GT-P2PRMI allows a host machine to instantiate a replicated service instance locally to provide closer service not only to itself but also to peer nodes in its vicinity.
- **Transparent Server Selection:** When a peer node makes an RMI invocation, GT-P2PRMI selects the server that might provide the better response latency (e.g., the one with lightest load and shortest round-trip time), and transparently directs the RMI invocation to the service instance at such a server node.
- **Transparent Failure Handling:** In GT-P2PRMI, when an RMI invocation returns exceptions (e.g., the peer running the service is overloaded and refuses to accept new requests, or the service has been terminated at this), GT-P2PRMI will retry on other peers until it finds one that can process the RMI request. From the peer nodes' perspective, the probability of service failure is much lower since it sees an exception only if all the peers that provide the service fail or refuse to accept the request.
- **Familiar Programming Abstraction:** GT-P2PRMI is an extension of traditional Java RMI and it provides all its functionality. Server and client programs that are aware of GT-P2PRMI can fully exploit its power while those that choose not to be aware of it can treat it like traditional RMI environment.

1.3.3 Thesis Outline

The reminder of this dissertation is organized as following:

Chapter 2 reviews the background and related work for this thesis. It includes a discussion of replication schemes and algorithms. It also discusses middleware frameworks that support caching and replication. We also review previous research work in multicast and overlay networks.

Chapter 3 describes AURA, an autonomous replication algorithm. This distributed algorithm uses probabilistic models and adapts to the size of the system. The algorithm

works especially well under heterogeneous load distribution because the algorithm allows more replicas to be created where the load is higher.

Chapter 4 and **Chapter 5** are dedicated to the discussion of update dissemination for object replication. Chapter 4 describes a preference-aware multicast grouping algorithm for disseminating data among a large set of end users using IP multicast. The algorithm considers both the available number of multicast channels and the bandwidth usage in access networks. Chapter 5 describes the method of building preference-aware overlay topologies that consider both overall overlay traffic and end-to-end network latency.

Chapter 6 presents GT-P2PRMI, a peer-to-peer based Java RMI middleware framework. In the chapter, we describe the implementation details of GT-P2PRMI and evaluate the performance of the implementation.

Finally, **Chapter 7** summarizes the contribution of our research and provides suggestions for future work.

CHAPTER II

BACKGROUND AND RELATED WORK

This thesis investigates the use of non-dedicated resources for supporting scalable distributed services. Specifically, this thesis studies peer-to-peer based object replication mechanisms to improve the scalability and performance of distributed applications. This chapter provides an overview of the background on the topics of the thesis and related work that has been done in the past.

2.1 Data and Service Replication

2.1.1 Overview

In the most general terms, replication refers to the use of redundant resources (e.g., software or hardware components) to improve the resiliency or performance of a system. Example uses of the replication principle include multiple processors in computer systems, RAID disks in storage systems, and multiple content servers for web services. In short, replication could be applied to almost every service.

In wide-area distributed applications, specifically the distributed services running over the Internet, content and service replication has been extensively used for many purposes. Replication can be used to improve service reliability and data availability under both system and networking failure conditions. It can be used to reduce the response time of remote service or improve the throughput of content delivery networks. It can also be used to improve the scalability of services that have a large number of users. In this section, we briefly review previous research that uses replication to achieve various goals in distributed systems.

Replication for Performance The use of replication has been proposed to improve the performance of services. In content delivery networks[53, 111, 21, 23, 107, 29] replication is an effective mechanism to improve the performance of the services. By replicating data to

the place where they are most requested, the content delivery throughput observed at the end users can be significantly improved.

Multiple servers are also used to improve the service response time in web hosting services[84, 85, 108, 50, 15, 27, 38, 36] and distributed data repositories[96]. Service response time could be affected by many factors, e.g., the load conditions of the servers, and the network latencies between the servers and the end users. Appropriately located servers that provide the same service can help to address both the load conditions and the network latencies.

CoopNet [79], a peer-to-peer content distribution scheme, helps servers tide over crisis situations such as flash crowds [93]. The authors focused on the application of streaming media content distribution. They addressed the challenge of flash crowd by redirecting clients' service requests to other clients who previously requested and cached the same content. Replication is also implicitly used in many other peer-to-peer systems, especially file sharing systems [3, 6, 2]. The peer-to-peer file sharing policies allow shared files to be stored where it is requested and serve for further requests from other peer nodes later.

Replication for Availability Replication is one of the most important resiliency strategies and has been used to increase the reliability of services and the availability of data in distributed systems[64, 14]. By providing multiple identical instance of the same data at different locations, the data can still be available when part of the system fails or goes offline. Replication for availability is an especially important principle in end-system based peer-to-peer networks, where the failures and loss of access happen frequently to the peer nodes. Matias et al. [14] showed that the availability of shared data in a peer-to-peer system can be improved from 24% to 99.8% at 6 times excess storage for replication. With the help of erasure coding, data can be highly available even when only a small subset of peers are online. Similarly, Farsite [9, 10, 39] implements a scalable, distributed file system that logically functions as a centralized file server but is physically built across a set of client desktop computers. The system monitors machine availability and places replicas of files on multiple client desktop machines to maximize effective system availability using different replication algorithms.

2.1.2 Replication Schemes

One of the most important problem in replication systems is replica placement optimization. Choosing the right replica placement approach is a non-trivial and non-intuitive exercise. Replica placement techniques include both passive caching[58, 63] and proactive replication[57], both centralized mechanism[84, 99, 53] and distributed methods[107, 57]. All approaches aim at meeting some latency and throughput related performance metric; some take such a metric as the direct goal of optimization, while the others use it as bounding constraints in problem formulation. The exact metric used to capture the performance goal differs among different approaches. For example, some systems aim at improving the average access latency[84, 99], while others guarantee that at least a certain percentage of accesses are served within a specified latency threshold. In general, deciding how many replicas to create and where to place them to meet a performance goal with minimal infrastructure cost is generally an NP-hard problem[54].

In this section, we give more detailed discussion of replica placement algorithms. Note that all replica placement approaches proposed in the literature are heuristics that are designed for specific system conditions. A heuristic that works well in one case may be completely inappropriate under different circumstances; it may be too costly or even unable to meet the performance goal when the context changes. Our interest mainly concentrates on schemes for improving service scalability and reducing the service response time through replication.

Centralized Replication Schemes Lots of work has been done to calculate the best locations for placing the replicated resources so that the average client-to-replica distance over all the participating client nodes is minimized. In these works, the proposed algorithms normally assume that they can collect the global knowledge about the end-to-end latency between all pairs of participating nodes and the pattern of request rates from all clients. The algorithms then make decisions in a centralized manner. For example, Qiu et al. proposed a greedy heuristic replication algorithm which aims at minimizing the average client-to-replica distance over all client nodes by computing the reduced average distance when greedily adding node into replica set one by one [84]. HotZone [99] presents a replica

placement algorithm that also tries to minimize the average client-to-replica latency. Different from Qiu’s work, the authors’ heuristic achieves this goal by clustering all nodes into several clusters first based on the network distances among them, and then picking one node in each cluster as a replica. HotZone relies on the accuracy of the geometric model of Internet latencies [43] to build the clusters. Radoslavos et al. proposed *max-router fanout placement* [80] which chooses the replica one by one in decreasing order of their node degree until all replicas have been chosen. It claimed a performance of 1.1 to 1.5 times of the performance of Qiu’s greedy replica placement algorithm [84].

Distributed Replication Schemes Although centralized algorithms can calculate the replica placement with high accuracy, the cost of collecting global information and disseminating centralized decision is also relatively high, especially when the scale of the system is large. Therefore, there are a number of researchers who have explored decentralized algorithms to minimize the average service response time over all the clients. Rabinovich et al. proposed a protocol suite for dynamic replication and migration of Internet objects [86] in which each node decides whether itself or its neighbor nodes should replicate the service to avoid overloading the existing service providers and meanwhile also avoid wasting resources. Similar to our replication algorithm presented in this thesis, this approach attempts to place replicas in the vicinity where a majority of requests are generated and meanwhile ensures that no servers are overloaded, by using a varying number of replicas in the system. Similar to our replication algorithm and different from the centralized ones discussed above, this work also takes into consideration the service processing time and queueing time at replicas. This replication algorithm, however, depends on the consensus of different nodes and assumes that the neighbor nodes will always agree to replicate the service when they are asked to. It also does not consider the situation where a replicating node can leave unexpectedly. Another decentralized replication algorithm by Ko and Rubenstein presented a fully distributed and self-stabilizing protocol that places replicated resources in a network of arbitrary topology such that the longest distance that a query must travel to find a particular copy of a resource is minimized [57]. It is similar to our replication algorithm in the sense that each peer node autonomously makes decisions about

whether it should replicate a service locally and provide access to it to others. Therefore, it is more suitable for a peer-to-peer environment. The authors also evaluated their protocol through thorough simulation which suggested that their algorithm has acceptable overhead cost and satisfying performance. However, their algorithm does not consider the processing time and assumes zero queuing time at service replicas, therefore it is not load adaptive.

Different replica placement algorithms have been addressed in peer-to-peer content distribution network [107, 100, 29]. Tang and Xu investigated the problem of placing object replicas (e.g., web pages and images) to meet the QoS requirements of clients with the objective of minimizing the replication cost [100]. Katz et al. presented a replica placement protocol, *dissemination tree* [29], that builds the content distribution tree on top of peer-to-peer location service PASTRY [88] while meeting QoS and server capacity constraints. The author addressed two crucial design issues. First, how to dynamically choose the number and placement of replicas, which is similar as our goal. Second, how to build the dissemination tree with small delay and bandwidth consumption. Instead of limiting ourselves to a tree topology, in our study, we try to construct general overlay topologies for disseminating data.

Other Work There are also a lot of work that tries to compare and evaluate the performance of different replication algorithms. The studies presented in [72, 33] evaluated the performance of uniform, proportional and square root replication algorithms in different network topologies including power-law random graphs(PLRG), normal random graphs, Gnutella graphs and two-dimensional grids through simulation. The simulation results showed that the square-root replication scheme is the best for query in unstructured peer-to-peer networks. Karlsson and Karamanolis described a method to assist system designers to choose a placement heuristic based on the cost of required infrastructure (e.g., storage capacity and network bandwidth), system topology, workload and performance goal [55].

2.1.3 Replication in Distributed Object Systems

Distributed object systems, e.g., SOAP [104], CORBA [78], DCOM [22], Java RMI, XML-RPC [5] gained popularity because of the great programming convenience they can provide.

They can hide the details of underlying OS and network systems. We are particularly interested in the caching and replication schemes available in these middleware frameworks. The caching framework implemented by Krishnaswamy et al. extended Java RMI framework by modifying the reference layer of the Java RMI stack [63, 60]. By allowing clients to cache the interested objects locally and access them locally, the remote method invocation response time can be improved, especially when the invocation requests pattern has a high ratio of reads to writes. Similarly, Eberhard presented requirements and mechanisms for efficient caching of objects for Java RMI applications [40]. Instead of caching the full object, his approach supports caching of reduced objects, which can save resources. Both of the above works addressed only caching objects and the cached objects are not accessible by other users. Filterfresh [19] is a Java package for building replicated fault-tolerant servers. It mainly implemented support for replicating *rmiregistry* at every node so that single-point-failures at the registry within the traditional Java RMI framework can be eliminated. Instead of replicating *rmiregistry* only to improve the fault tolerance, we replicate the objects to both handle failures and reduce invocation latency. Globe [103, 17] designed *distributed shared object* which includes replication *subobject* that can encapsulate its own policies for replication, migration and so on. Its goal is to support replicating different objects with different replication schemes. Our work is different in the sense that we not only support replicating distributed objects, but also transparently handle invocation failures and resend invocation requests. Fragmented objects, like Globe’s distributed shared objects, are physically distributed across multiple machines, encapsulating their own distribution policy. However, fragmented objects do not explore how dynamice and fair resource utilization can be ensured at peer nodes. In the W3Objects system [34], web resources are encapsulated into distributed objects that can have their own replication scheme. The model is strongly based on the notion of remote objects, but it is less flexible than a model in which objects can be truly physically replicated and distributed.

2.2 Peer-to-Peer Systems

2.2.1 Peer-to-Peer Technologies

Different peer-to-peer technologies have been used to utilize both the storage and CPU resources available in peer nodes to improve the performance of different applications. Centralized peer-to-peer system (e.g., Napster [3]) have a central repository that stores an index to all files available in the system. Distributed peer-to-peer network, which can be categorized into structured and unstructured, search replicated files in a distributed fashion by forwarding the query to the overlay neighbors instead. Structured peer-to-peer networks (e.g., Chord [98], CAN [90], Pastry [88], Tapestry [116]) use a distributed hash table to look up files. This scheme is useful for locating files with exact name, but cannot perform well when given partial matching keyword. Unstructured peer-to-peer networks (e.g., Gnutella [6], Kazaa [2]) uses scoped-flooding search, therefore, performs well with partial matching keyword and had become very popular. Seti@home [4] is a good example where distributed computation cycles available on personal computers can be collected to run applications that otherwise require expensive and powerful super-computers. Peer-to-Peer network has also been used for media streaming where peer nodes can act as media servers temporarily [79, 59].

2.2.2 Peer-to-Peer Middleware

Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. It is key to openness and interoperability. Middleware masks differences between OS platforms and networks and makes the communication between information consumers and suppliers easier.

Some of the distributed object middleware framework target general support for distributed applications in a dynamic environment. The P2PS middleware [76] provides the developer with a means for building and working with P2P overlay applications, offering various primitives and services such as group communication, efficient data location, and dealing with highly dynamic networks. P2PS implements Tango [25], an efficient algorithm

for constructing structured P2P systems. The main functionality provided by P2PS includes network management primitives such as create, join and leave a network, communication primitives such as one-to-one, broadcast and multicast, and monitoring primitives.

Omnix [65] provides an abstraction between the application and the underlying network by providing higher-level functionality such as distributed P2P searches and direct communication among peers. The central idea of this work is to apply the well-established ISO OSI layered model on P2P middleware systems. Therefore, it consists of three layer: the *Transport Layer*, the *Processing Layer* and the *P2P Network Layer*.

JXTA[49, 102] protocols establish a virtual network overlay on top of the Internet, allowing peers to directly interact with each other independent of their network connectivity and domain topology (firewalls or NATs). Project JXTA enables application developers, not just network administrators, to design network topology that best match their application requirements. Multiple ad hoc virtual networks can be created and dynamically mapped into one physical network. The goal of JXTA is to have a billion network services, all addressable on the network, to discover and interact with each other in an ad hoc and decentralized manner through the formation of a multitude of virtual networks. The JXTA virtual network standardizes the manner in which peers discover each other, self-organize into peer groups, discover network resources, communicate, and monitor one another. Although JXTA can support large number of peers to access large amount resources on Internet, the performance of underlying implementations is not optimized, therefore, the response time for using services could be long.

2.3 Advanced Network Technologies for Group Communication

2.3.1 Overlay Networks

In overlay networks, specially designed nodes distributed over the Internet form another layer on top of the underlying Internet routing substrate. The nodes cooperate with each other to forward data on behalf of any pair of communicating nodes in the overlay network; each node does not only generate and sink but also forwards traffic. Researchers have studied

the use of overlay networks in various areas. Overlay networks based on end-systems have been used to implement application layer multicast [31, 18, 81, 71]. Infrastructural overlay networks have been proposed to circumvent BGP faults and constraints [12, 92], provide countermeasures to DoS attacks [56], and support Quality of Service [67, 69]. They have also been used for content distribution [59] and naturally act as the communication method for peer-to-peer file sharing systems(e.g., [7, 32]).

Recently, there is considerable interest in the use of overlay networks as a general solution to add more flexibility and control to the routing infrastructure [83, 101, 44, 20] of the Internet. While the Internet has been fully commercialized and evolved into a ubiquitous medium of communication, its fundamental routing infrastructure cannot easily be changed. Overlay networks can be used to provide testbeds for new technologies [46, 41, 82] and facilitate the development of new network technologies.

The *overlay topology* is one of the most prominent configurable components of an overlay network. An overlay topology constructed in favor of the underlying networks and the upper application requirements can significantly improve the performance of the whole system. For example, to reduce the latency of overlay networks, researchers have considered network proximity when building overlay topologies [105]. Specially, Distributed Binning [89] uses landmarks to calculate distance between overlay nodes, puts close nodes into the same bin, and forms a low-latency overlay network by using half of the overlay links for intra-bin connection and another half for inter-bin connection. Our methodology is similar to theirs. However, they did not consider the preference heterogeneity and did not address the issues

Tree and mesh topologies for overlay multicast have been extensively studied [31, 18, 81, 71] in the literature. However, we are interested in overlays with more general communication specifications that are defined with preference matrix and we do not limited ourselves to only tree or mesh topologies.

2.3.2 IP Multicast and Application Layer Multicast

Multicast is an efficient way to deliver the same information to a group of destinations simultaneously. Instead of sending multiple copies of the same information, one copy for each

destination, the sender sends the information once and the multicast protocol is responsible for delivering the message to all destination nodes.

Multicast can be implemented in any layer of the network stack. But traditionally multicast function has been mainly implemented in the network layer[42, 16], e.g., IP layer. Over a decade, however, IP multicast has been facing various difficulty in its deployment over the scope as wide as the Internet. Over the past a few years, though still keeping IP multicast as an option, researchers have also seek to move the multicast functionality into the application layer [28, 112, 47, 51, 35].

Application layer multicast protocols can be can be classified into two classes: tree-first approach or mesh-first approach. Tree-first approach directly builds an overlay tree topology out of the physical networks (e.g., Yoid [47] and ALMI [35]) and use the tree as the multicast data paths. Mesh-first approach first constructs a mesh on top of the physical networks for maintenance purpose and then generates a source-specific tree over the mesh topology as the multicast data paths, e.g., Narada [112] and Scattercast [28]. Most application layer multicast protocols form overlay networks over end-systems, so application layer multicast is often also termed as end-system multicast.

2.3.3 Preference-Aware Communication

Event-based middleware system, i.e., the publish/subscribe system like Gryphon [118, 117], JEcho [119, 120] and PADRES deliver information from a set of producers to a set of interested consumers. A set of broker machines are deployed to form a static overlay network that routes information from producers to consumers. Gryphon [118] is based on an information flow model and implements publish/subscribe event delivery with JMS interface [1]. It also includes several extensions for event delivery, including guaranteed delivery, durable subscriptions and relational subscription [52][94][95]. Similar to Gryphon, Siena [8, 26] delivers events through an overlay network of brokers. Subscriptions are conjunctions of predicates over the event attributes. Several broker network topologies are proposed in Siena. Reverse path forwarding is used for broadcast routing and subscription filtering takes place on the data delivery paths.

Instead of deploying dedicated brokers to route data, some research work let participating peer nodes form an overlay and forward data for each other [119, 120]. Therefore, it can provide dynamic reconfiguration of application-layer networks. This is better than the above systems which have been originally built for static environments and assume a static event dissemination structure. Having fixed event dissemination structures makes them difficult to adapt to changing network conditions, hence unsuitable for applications in dynamic peer-to-peer environments where nodes can join and leave unexpectedly.

Compared with event-based middleware systems, multicast provide more general group communication support for disseminating data based on users' preference. Multicast grouping is an important issue when multicast is used for data dissemination. Multicast grouping problem is not new. Researchers have studied this problem in distributed interactive simulation applications and proposed cell-based and entity-based grouping algorithms [87] [68] [106] [24] [77] [74] [75]. Their efficiency has been extensively investigated in [121] [73]. The algorithms are simple yet powerful in simple gaming environment, where the relationships between game entities can be defined by simple vision domain rules. However, they might not be flexible enough for more complex gaming environments or other distributed interactive applications and therefore may not be suitable for a general distributed object platform.

Wong et al. were the first to investigate the multicast grouping in a general form and proposed an algorithm based on the k -means clustering method [110]. It proposes a k -means clustering algorithm that cluster users based either on the distance among sources or distance among receivers. This algorithm does not serve our purpose for two reasons. First, the algorithm targets at optimizing the overall traffic received by all receivers. The heterogeneity of receivers is ignored, therefore the algorithm may give a solution that overloads certain receivers. Second, the algorithm considers either only similarity among sources or receivers, while the similarity between connections could be a more general and flexible criteria. This limits its scope of finding better grouping. Our work addressed this problem by defining the distance among source-receiver pairs.

2.4 Summary

GT-P2PRMI, a distributed object framework, supports dynamic invocation in peer-to-peer environment which is not addressed in most of the above related work. The updates dissemination schemes using multicast and overlay network distinguish themselves with their preference-aware features since most work does not consider users' heterogeneous preferences. In the next several chapters, I will address each component of our work in details.

CHAPTER III

AURA: AUTONOMOUS REPLICATION ALGORITHM

In this chapter, we present AURA, an autonomous replication algorithm that is used to decide the placement of object replicas so that the average service response time can be minimized under different load distribution. First, we state the four practical goals of our replication algorithm for peer-to-peer environment. Second, we formalize the replica placement problem. Third, we describe our algorithm that achieves all the four goals we stated. Last, we present the evaluation results from both simulation and actual experiment with the Planetlab testbed.

3.1 *Introduction*

In this chapter we propose a new scheme for object replication in distributed object systems. The scheme avoids replicating or caching object states at every peer node so that the consistency overhead can be managed. However, since the scheme is peer-to-peer based, any peer node can potentially act as a replicating server. Instead of being replicated at dedicated servers, the objects are replicated at a *fraction* of the peer nodes, who in turn serve the requests from other peer nodes in their *vicinity*. Compared with a scheme caching or replicating objects at every peer node, such a scheme has more control and tunability in the consistency maintenance overhead. Compared with a scheme using fixed dedicated replication servers, such a scheme is more scalable and adaptable in terms of resource provisioning and is more flexible and tunable in terms of replica placement.

There are several challenges, that remain unaddressed, particularly in peer-to-peer environments, where global knowledge is usually not available to the replication algorithms. Firstly, how to decide the placement of server object replicas based on the distribution of service requests? The placement should be dynamically adjusted when the distribution of service requests changes. Secondly, how to maintain the number of peer nodes that are

concurrently replicating the server objects at a desired level, e.g., a certain percentage of total number of peer nodes? A degree of replication that is too high may cause excessive consistency maintenance cost. Too few replicas may not provide enough service capacity, therefore causing overload at the replicating nodes. Thirdly, how to achieve fairness among the peer nodes in terms of their contribution to the system? Fairness is a basic principle of resource sharing and is a major consideration in many other peer-to-peer based systems.

We focus on solving the above problems and propose a dynamically adaptive object replication scheme for scalable distributed services. The key features of the schemes are following:

- Each peer node makes its replication decisions based only on its locally maintained information.
- The algorithm automatically and continuously adapts to changes in the number of nodes and service request rate in the system.
- The algorithm is fair that each peer node contributes a similar amount of time in serving other nodes.

In our replication schemes, we made the following assumptions. First, we assume that the participating peer nodes are not selfish or malicious. They honestly follow the specified protocols unless they fail. When they fail, they fail quietly without generating any input to the system.

Second, we do not limit ourselves to a specific suite of consistency protocols. Instead, we are most interested in the update dissemination algorithms that can be used by any consistency model and protocol to disseminate updates. Such protocols can easily provide best-effort type of consistency.

Third, in our replication algorithms, we assume that the total amount of computation resources provided by a fraction f of peer nodes is sufficient to serve all the service requests generated by all the peer nodes. We also assume that every peer node contributes similar amount of resources, i.e., any replica set consisting of a fraction f of all the peer nodes provides similar amount of total computation resources.

3.2 *Peer-to-Peer Based Object Replication*

Figure 2 shows the use of peer-to-peer based object replication to reduce the response time of remote calls and improve the performance of distributed object systems. To reduce the communication time involved in the processing of the remote calls, a *portion* of peer nodes replicate the service objects from the owner server node. Each of the peer nodes replicating the service objects in turn handles requests from other peer nodes in its vicinity that do not have service instances. Meanwhile, the replicating peer nodes cooperate to make sure that the state of an object is consistent among all replicas of the object based on a specified consistency policy. Since the peer nodes are not dedicated server nodes, the portion of peer nodes that replicate the service can change with time. As shown in Figure 2, some of the peer nodes that were previously replicating the service at time I have deleted the service by time II; and some peer nodes that were not replicating the service at time I have started new replicas by time II.

In this algorithm, we focus on the issues of replica placement, i.e., the selection of a certain number of replica locations from the set of peer nodes and omit the discussion about how to maintain the consistency between the owner server node and the set of replicating peer nodes — much of the consistency maintenance mechanisms and protocols proposed in [60] can be used here. The replica placement problem is NP-hard because of its combinatorial solution space. Similar problems have been studied in the context of web service and content delivery network with a focus on performance optimization, usually assuming the availability of global knowledge and central control [113]. In the context of peer-to-peer based object replication, however, the global knowledge and central control are usually not available, and distributed algorithms are needed. Besides, apart from pursuing performance, the replica placement algorithm should also consider the dynamic conditions of a peer-to-peer environment as well as fairness among peers. Specifically, we are interested in finding replica placement algorithms that address the following issues.

3.2.1 Performance

The goal of object replication in distributed object systems is to reduce the impact of networking communication latency on the response time of remote calls. Assume there are n peer nodes $U = \{u_1, u_2, \dots, u_n\}$ in the system. The object replicas are placed at a subset of the peer nodes, denoted by S . Each peer node u_i selects the *nearest* replicating peer node that is not overloaded in set S as its local server, denoted by $\mathcal{L}(u_i)$, and directs all its remote calls to this server. Assume that at time t , the peer node u_i is generating remote calls at rate r_i requests per unit of time, then a replica placement S with good performance should be one that generates a low value for the following cost metric:

$$\sum_{i=0}^{|U|} d(u_i, \mathcal{L}(u_i)) r_i \quad (1)$$

where $d(u_i, u_j)$ is the network latency between nodes u_i and u_j . For simplicity of presentation we assume the network latencies between two nodes are symmetric, i.e., $d(u_i, u_j) = d(u_j, u_i)$.

3.2.2 Controlled Number of Replicas

It is straightforward to see that the cost in Eq. 1 is minimized if every peer node replicates (i.e., $S = U$). However, as we previously discussed, such a replication scheme incurs large consistency maintenance overhead and forces the system to adopt a lower level of consistency. A good choice for the number of replicating nodes (i.e., the size of S) is a tradeoff between the consistency overhead and performance needs. Too few replicas cannot effectively reduce the response time of remote calls. Too high a degree of replication may not necessarily improve system's responsiveness significantly because of excessive consistency maintenance cost. The problem of deciding the appropriate number of replicas in the system is beyond the scope of our work. Instead, we assume the number of replicas is given as a certain fraction of the total number of peer nodes (i.e., $\frac{|S|}{|U|} = f$) in the system and we are most interested in how to maintain the number of replicas at this desired level. The reasoning behind the use of a fraction instead of a fixed number is that the system can be dynamic in terms of the number of peer nodes and it is desirable that the number of replicas

in the system adapts to the size of the system.

3.2.3 Autonomous Adaptability

In a peer-to-peer system, the set of peer nodes can change over time. The system should be able to adapt to the changes dynamically. In a system where global knowledge about the system is available and central control is used, the system can keep recomputing the placement of the replicas based on the global knowledge, and coordinate the adjustment in replica placement using the central controller. In a peer-to-peer environment, however, we are most interested in autonomous and distributed recomputation and adjustment to the replica locations without relying on a central controller. Such autonomous adaptability is especially desirable and potentially effective in an environment where each peer node could make efficient use of the information it can learn from nodes in its neighborhood.

3.2.4 Fairness

In peer-to-peer systems, fairness among peer nodes is usually an important design goal. For example, in many peer-to-peer file sharing systems, special mechanisms are used to ensure that every peer contributes a reasonable amount of upload bandwidth to the system in order to get higher download bandwidth [6]. In general, however, the definition of fairness can vary in different types of systems, with focus on different resources such as the usage of network bandwidth, contributed CPU cycles, or the number of exchanged files. In the context of peer-to-peer based object replication systems, we are most interested in the fairness that is based on the number of calls made and serviced by a peer node over some interval of time. Every peer node could keep generating remote calls, but at each single moment only a portion of the peer nodes are replicating the service objects. It is desirable that different peer nodes replicate the objects at different times and each peer node provides a fair overall amount of service.

Even in this specific context, the exact definition of fairness can vary. In a peer-to-peer based object replication system, the term fairness could have one of the following meanings:

- *Service Time Fairness:* In a time window of our interest, every peer node serves as the replication server for approximately the same amount of time.

- *Service Load Fairness*: In a time window of our interest, each peer node serves equal number of requests.
- *Credit-Debit Fairness*: In a time window of our interest, every peer node serves the same number of requests as it generates during the same period.

All these definitions reflect valid design concerns and deserve in-depth discussion. In this work, however, we focus our discussion on service time fairness, and leave credit-debit fairness and service fairness as future work.

3.3 AURA: An Autonomous Replication Algorithm

In this section, we present AURA, an autonomous replication algorithm for peer-to-peer based object replication systems. The algorithm is designed to address all the issues discussed in Section 3.2. First, the algorithm is autonomous. Each self-managing peer node makes its own decisions about service replication, using only the information it collects from its neighborhood. The capability of making decisions autonomously using only locally maintained information is key to scalable coordination in systems with potentially a large number of peer nodes. Second, the algorithm is probabilistic and adaptive. Each peer node’s replication decisions are probabilistic. Although each peer node makes its own decisions, the overall effect of their probabilistic behavior is that the system maintains a certain number of replicas and adapts this number when peer nodes join and leave. New replicas are added when and where they are needed but the total number of replicas in the system is controlled. Third, the algorithm is fairness-aware. Each peer node keeps track of its contribution to the system and adjusts its behavior accordingly to ensure it contributes a fair share of service time to the system. Finally, performance is still an important goal of AURA. Generally, some degradation in performance is usually unavoidable when an autonomous algorithm relying on incomplete knowledge is used instead of an algorithm relying on global knowledge. Furthermore, enforcing fairness among peer nodes also limits the performance to some degree since even the set of peer nodes that provide the best performance should not run more than their share required by the fairness. AURA algorithm uses information obtained from periodic messages that are exchanged among neighbors for the coordination

among nearby nodes to reduce the performance loss caused by not only the lack of global knowledge and central control but also from the need to maintain the fairness constraint.

To explain the intuition behind AURA algorithm, we first examine the whole spectrum of replica placement algorithms, in terms of the level of coupling among different nodes' actions in these algorithms. At one extreme of the spectrum are a class of algorithms that rely on the availability of global knowledge. In this class of algorithms, each peer node's decision making depends on the states of all the other peer nodes. For example, all the peer nodes send their information to a central controller which can integrate the information and send the results back to all the peer nodes for them to make decisions. At the other extreme are a class of algorithms in which each peer node makes its decision purely based on its own state. This eliminates the communication overhead for a peer node to learn the states of other nodes. Actually, by introducing probabilistic behavior to each node, this class of algorithms can achieve some of our design goals with very little overhead — statistically, if each peer node randomly selects a fraction f of its time and replicates the service during that period of time, the average replica fraction in the system, as a function of the number of nodes, should also be approximately f , which meets both the goal of achieving fairness in service time and the goal of maintaining a stable replication fraction f . However, the complete lack of coordination with other nodes in decision making often results in replica placement that does not match well with the distribution of nodes and their loads in the system. This causes relatively high service response time and degraded performance.

The AURA algorithm avoids the two extremes. Instead, it chooses a point between the two extremes. Similar to the previously discussed algorithms in the second class, the operation of AURA largely involves probabilistic behavior to determine when a peer node should have a replica of the service. However, instead of making decisions totally independent of other nodes' states, each peer node uses the information obtained from nearby replicating nodes to prevent unnecessary replicas from being created. Different ways can be used to collect information about nearby replicas. We used two methods that do not need centralized control and collect such information through distributed fashion in our work. The first way is using local query, which can be used to collect the replica fraction in the local area,

the location and the load condition of nearby replicas. The second way is using heartbeat, which can be used to learn the location and load condition of nearby replicas and even to estimate the global replica fraction in the whole system. The use of query or heartbeats increases the protocol costs but also increases the performance of the system. The details of the two protocols to collect replicas' information will be described later.

As we discussed in Section 3.2.4, fairness can be defined in several different ways depending on the applications' specific requirements. To meet different fairness definitions, however, the algorithm needs to act differently. In our work, we concentrate on service time fairness.

In AURA algorithm, each peer node is always in one of the three states: *replicating*, *sleeping*, and *ready*. The possible transitions among the three states are shown in Figure 3. When a peer node is in the *replicating* state, it is replicating the service object and can serve other peer nodes' requests. During the replicating state, a peer node makes replica deletion decision based on either its served request number or the pre-assigned time slice. After a peer node shutdowns its replicated service, it enters the sleeping state. When a peer node is in the sleeping state, it does not make replication decision, therefore is impossible to enter replicating state from sleeping state. After the sleeping state term is over, the peer node will enter the ready state. When a peer node is in the ready state, it is not replicating the service yet but is ready to start replicating the service if needed. A peer node makes replica creation decision only when it is in the ready state.

Timing of Replication Decision In AURA, the frequency with which a node makes replication decision affects its protocol overhead, its responsiveness to changing system conditions, e.g., when nodes join and leave the system, or when replicas are created and deleted, and also the fairness requirement. If the nodes query and evaluate the current replica fraction in their vicinity aggressively, the replica fraction in each local area can be maintained at target value f accurately even when the number of nodes in the system changes quickly. But this also incurs more communication overhead in the overlay network. If a node queries and evaluates the current replica fraction in its vicinity lazily, the protocol overhead of AURA is lower but the number of replicas in the system may not catch up with

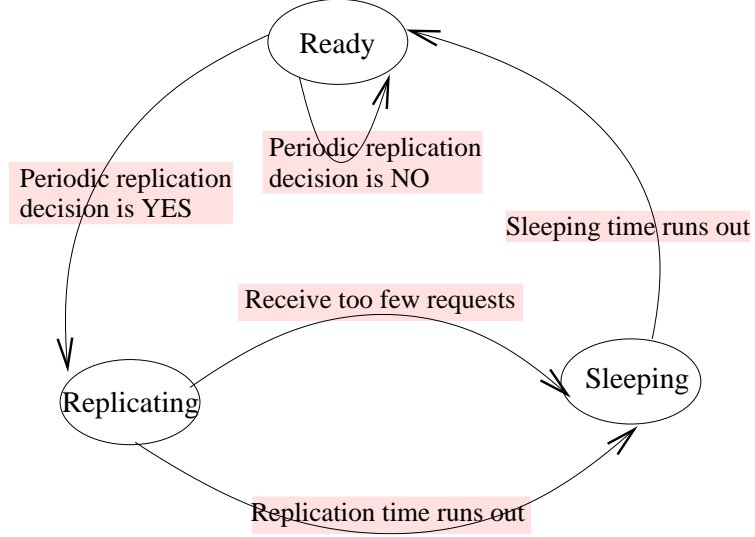


Figure 3: State Transition Diagram

the total number of nodes in the system.

The timing with which a node makes replication decision also affects the fairness requirements. There are various ways for a node to decide when it should recompute its replication decisions. For example,

1. Make decision every L requests
2. Make decision every T interval

Intuitively, if the definition of fairness requires the nodes that generate more requests to serve more requests from others, the first rule is more desirable. If the definition of fairness requires every node to serve similar amount of time, the second rule is more desirable. In our thesis, we concentrate on the definition of fairness that requires each node to contribute similar amount of service time, therefore, we let each node make its replication decision periodically. If we want to implement the credit-debit fairness definition, the algorithm should be extended to let each node make its replication decision upon each generated request (or every L requests). We did not implement the credit-debit fairness requirement in our algorithm, but this could be easily extended to do so.

Below, we will first talk about the details of the heartbeat protocols we used, then we will describe how a node periodically makes replication decision. At last, we will explain

Notations	Meaning
f	target replica fraction
m_i	counter value of node u_i during the counting protocol
\mathcal{T}_i	total recording time of node u_i (sliding window size)
α_i	total replication time in the past sliding window of node u_i
β_i	total missed replication in the past sliding window of node u_i
τ_i	last replication time
ϕ_i	sleeping ratio of node u_i
γ_i	current servicing load of node u_i
μ_i	load upper bound of node u_i
$\mathcal{R}(u_i)$	set of replicas in node u_i 's vicinity

Table 1: Parameters Used in Algorithm

how the parameters are dynamically adjusted to meet our goals. The parameter used in this algorithm are summarized in Table 1.

Heartbeat Heartbeat is commonly used in overlay network to keep connectivity [88, 30]. Basically, each peer node periodically sends out heartbeats to its directly connected neighbors to inform them that it is alive. In this section, instead of using query messages, we use heartbeats to obtain some more information than just connectivity. The major benefit of heartbeat is that it incurs less overhead than query.

First, heartbeat is used to advertise replicas' information including both location and load. The load information includes a replica's current servicing rate and its servicing upper bound decided by its contributable resources. The location and load information of each replica is tagged with a TTL and therefore is only included in other peer nodes' heartbeats if its TTL has not been decreased to zero. Therefore, each replica is only visible to other nodes within a neighborhood of TTL hops. Through the heartbeat message, each peer node, regardless of its replicating status, can learn the number of replicas in its vicinity, and also the location and load information of each of those replicas.

Secondly, heartbeat is used to obtain the approximate replica fraction in the system. AURA uses a distributed counting protocol to exchange information among direct neighbors to achieve a global estimate of the current replica fraction among all peer nodes currently in the system. The protocol runs every phase with length of P time units (a span of P time units is called a phase) and each node starts the protocol at the beginning of each

phase. For example, we can define the starting time of each phase as 1:00, 1:10 At the beginning of each phase, each peer node u_i initializes its counter m_i as follows: if u_i is a replica, m_i is equal to 1, else m_i is equal to 0. After that, at each odd round of heartbeat, each peer node sends its current counter value to its direct neighbors; at each even round of heartbeat, each peer node updates its counter m_i following the algorithm shown in Algorithm 1. The basic idea is to shed the value from nodes with higher counter values to those with lower values. Eventually, the counter value m_i of each peer node u_i will converge to the replication ratio of the system. The accuracy of m_i is affected by the length of a phase, P . We observe that a small sacrifice in accuracy can significantly reduce the time required for converging. Our simulation shows that a system with one thousand nodes can quickly reach the state where more than 90% of the peer nodes currently in the system reach the range of $[0.9 * f_{current}, 1.1 * f_{current}]$, where $f_{current}$ is the current actual replica fraction in the system.

Algorithm 1 Actions for Peer Node u_i to Split m_i Among Direct Neighbors

```

 $m'_i = 0;$ 
 $S$  is the set of nodes that are  $u_i$ 's direct neighbors;
 $sum = m_i + \sum_{u_j \in S} m_j;$ 
 $count = 1 + |S|;$ 
 $avg = sum / count;$ 
if  $m_i < avg$  then
     $m'_i = m_i$ 
     $b_i = 0;$ 
else
     $b_i = (m_i - avg) / count;$   $m'_i = avg + b_i;$ 
end if
SEND  $b_i$  to all direct neighbors
 $m_i = m'_i + \sum_{u_j \in S} b_j ;$ 

```

Creating Replicas Each peer node u_i , when in *ready* state, periodically makes replication decision based on the load information of replicas in its vicinity and the resources it has contributed in the past. The creation decision should not only promote the fairness but also adapt to load distribution to improve the performance. Therefore, the creation decision is affected by two factors.

First, the more overloaded the replicas in a peer node u_i 's vicinity are, the higher

probability peer node u_i should create a replica. In that way, more replicas can be created in areas with higher request rate. Since each replica heartbeats its replicating status, including request serving rate and request serving upper bound, to its vicinity within TTL hops, each peer node u_i , regardless of its replicating status, can learn the set of replicas in its vicinity, denoted as $\mathcal{R}(u_i)$, the request serving rate γ_j and request serving upper bound μ_j of each peer node $u_j \in \mathcal{R}(u_i)$, through the heartbeat messages. The average request serving rate $\bar{\gamma}_i$ is the average of the request serving rates over all the replicas in a peer node u_i 's TTL-scoped vicinity. γ_j/μ_j reflects the overload condition of peer node u_j , and the value of 1 means u_i is fully loaded. The average overload ratio $\bar{\theta}_i$ is the average of the overload conditions over all the replicas in a peer node u_i 's TTL-scoped vicinity. They are defined as follows:

$$\bar{\gamma}_i = \frac{\sum_{\gamma_j \in \mathcal{R}(u_i)} \gamma_j}{|\mathcal{R}(u_i)|} \quad (2)$$

$$\bar{\theta}_i = \frac{\sum_{\gamma_j \in \mathcal{R}(u_i)} \gamma_j / \mu_j}{|\mathcal{R}(u_i)|} \quad (3)$$

Secondly, to promote fairness, the less resources a peer node u_i has contributed in the past, the higher probability it should contribute its resources to create a replica. Ideally, each peer node should contribute a fraction f of its time to replicate a service. But since the replication decision is affected by the status of other replicating nodes, a peer node may not contribute enough replication time in the past. Therefore it should try to compensate for the replication time it has missed by contributing more resources in the future. To achieve this goal, we require each peer node u_i to record the time it has contributed to replicate the service, denoted as α_i , and the time it has spent in recording its replication time, denoted as \mathcal{T}_i , in its local memory¹. Then peer node u_i can calculate the missing time $\beta_i = f\mathcal{T}_i - \alpha_i$ when it needs to make creation decision. The ratio $\frac{\beta_i}{\alpha_i + \beta_i}$ reflects how much a peer node u_i has missed its replication time in the past. The higher this ratio is, the higher probability a peer node u_i should create a replica locally.

¹The values of α_i and \mathcal{T}_i are decreased periodically. For example, \mathcal{T}_i is decreased by $\mathcal{T}_i/2$ and α_i is decreased by $f\mathcal{T}_i/2$ periodically. Otherwise, as the value of \mathcal{T} and α_i builds up to a really large value, the value of $\frac{\beta_i}{\alpha_i + \beta_i}$ becomes less meaningful.

We combine the two factors discussed above and get the creation probability p_i with which peer node u_i should create a replica, as follows:

$$p_i = \bar{\theta}_i \frac{\beta_i}{\alpha_i + \beta_i} \quad (4)$$

When all the replicas in peer node u_i 's vicinity are overloaded and $\alpha_i \ll \beta_i$ and all nearby replicas are overloaded, the creation probability p_i is equal to 1; when a peer node u_i has contributed enough resources, the creation probability p_i is equal to 0.

If the replication decision is yes, u_i sets its replicating time as β_i , its missed replication time, and enters the *replicating* state. If the decision is no, u_i does not create a replica but stays in the *ready* state. It will try to create a replica in the next period when making the replication decision with updated $\bar{\theta}_i$ and $\frac{\beta_i}{\alpha_i + \beta_i}$ again.

Deleting Replicas After a peer node u_i creates a replica locally, it can serve requests from other peer nodes. If the replica created at peer node u_i does not receive a lot of requests during its serving time, it implies that this replica may not be created at a useful location therefore better be deleted. The replica at peer node u_i will be deleted when

1. the scheduled time slice β_i runs out;
2. u_i receives much fewer requests than other replicas in its vicinity: $\gamma_i < \gamma_j, \forall u_j \in \mathcal{R}(u_i)$ and $\gamma_i < \bar{\gamma}_i/2$.

If the replica is deleted due to the second condition, the actual amount of time peer node u_i replicates the service, denoted as τ_i , will be smaller than the scheduled time β_i , i.e., $\tau_i \leq \beta_i$. After deleting the replica, u_i will enter the *sleeping* state and stay in the sleeping state for $\tau_i \phi_i$ units of time. ϕ_i is ratio between the time u_i will stay in sleeping state and the time u_i has just stayed in the replicating state. The sleeping ratio ϕ_i is tunable by each peer node u_i itself. The goal of tuning the sleeping ratio ϕ_i is to control the number of replicas in the system to reach the target replica fraction f in the system. We will describe how to tune this parameter later in this section. In the sleeping state, peer node u_i will not try to create a replica under any condition.

Tuning Control Parameter If the creation probability of each peer node u_i is always 1 and each replica is deleted at the end of its scheduled time slice β_i , sleep ratio $\phi_i = \frac{1-f}{f}$

is enough to maintain the target replica fraction of f in the system. However, the value of the creation probability p_i and replication time τ_i are affected by other replicas' status therefore vary during different time and under different load distribution.

AURA uses a feedback-based tuning mechanism to adjust the sleep ratio to an appropriate value. Each peer node u_i adjusts the value of its sleeping ratio ϕ_i dynamically based on the estimated replica fraction, m_i , obtained from the distributed counting protocol we described earlier in this section. Initially, ϕ_i is set to $(1 - f)/f$. Periodically, based on the observed replication ratio m_i , which approximately reflects the global replica fraction, node u_i adjusts ϕ_i as follows: if $m_i \leq 0.9f$, ϕ_i is decreased by 1; if $m_i \geq 1.1f$, ϕ_i is increased by 1; otherwise, ϕ_i is kept the same.

3.4 *Performance Evaluation*

In this section, we evaluate the performance of AURA for large scale systems through simulation. We also conduct some experiments with the PlanetLab test bed [82] to measure the performance of our replication algorithm in an environment with realistic end-to-end network latency, real network traffic and shared load with other users on the same node.

3.4.1 *Simulation Setup*

We build a discrete event driven simulator to measure the performance of AURA. The simulation takes the following parameters: (1) A group of N nodes and the network communication latency between every pairs of nodes; (2) Requests generated following exponential distribution with parameter λ ; (3) Targeted replication fraction f . The overlay network topology is generated as follows. N nodes are randomly distributed in a m dimensional virtual space. The end-to-end latency between any pair of nodes is the euclidian distance between the two nodes in the virtual space, as GNP does [43]. We build an overlay topology with an average degree of 5 similar as [57]. The 5 overlay neighbors of a peer node are randomly chosen from its 10 nearest nodes². This is a simple way for constructing

²We use the connected topologies and discard the partitioned ones.

a topology-aware overlay network but not the best one. There are more complicated algorithms for building a topology-aware overlay [89, 66] and they should result in better performance. To avoid the impact of different overlay construction algorithms, we decided to use this simple random one. All the results in this section are based on overlay topology with 1000 overlay nodes. The service object is initially replicated at 5% (50) randomly selected nodes. During the simulation, each request is directed to the closest unloaded service replica.

3.4.2 Stability of Replica Fraction

Figure 4 and Figure 5 plot how the number of replicas in the system varies with time. x axis represents the simulation time, y axis represents the number of replicas. We sample the results every 10,000 time units and plot the figures. The sleep ratio ϕ_i is fixed in Figure 4 and adjustable in Figure 5. Figure 4 shows that after an initial adaption period, the number of replicas in the system reaches a dynamic equilibrium and the reached equilibrium is lower than the target of 5% when ϕ_i are fixed at 19. In Figure 5, sleep ratio ϕ_i is not adjustable from time 0 to 50 and adjustable by each peer node itself from time 50 to 450. We can see that after each peer node starts to adjust ϕ_i based on its estimated replica fraction, the average number of replicas starts to rise, and it reaches its stable target range eventually.

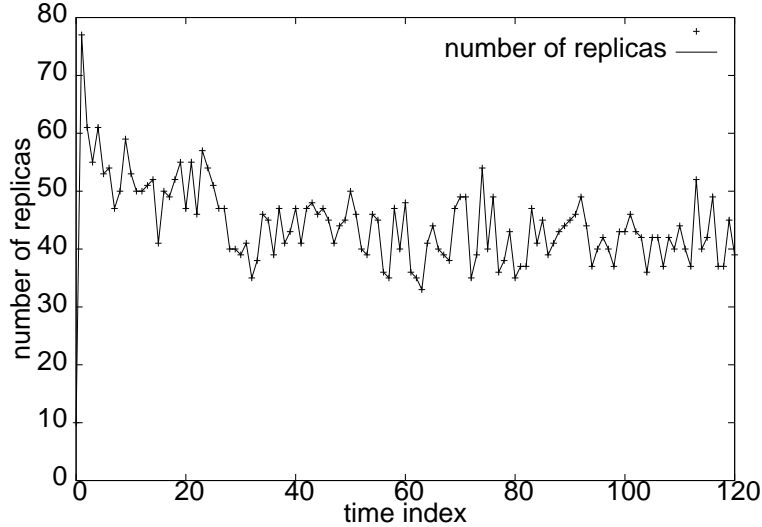


Figure 4: Variation of the number of replicas in the system when sleep ratio ϕ_i of each node u_i is fixed

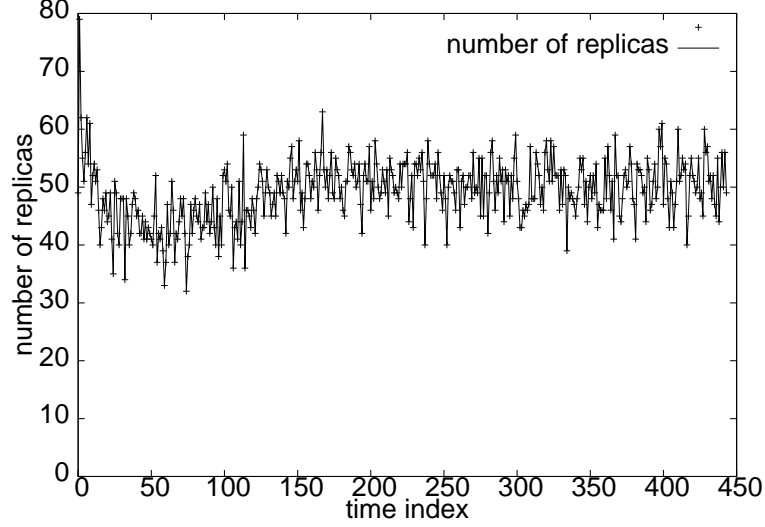


Figure 5: Variation of the number of replicas in the system when sleep ratio ϕ_i of each node u_i is adjustable

3.4.3 Fairness of Resource Contribution

Figure 6 plots the total amount of time each peer node spends in replicating the service. The x axis represents the index of each node, the y axis is the sum of all the time when a node is replicating the service. The simulation runs for 4,500,000 time units and there is no join or leave during the simulation. This figure shows that each peer node contributes its resources for approximately similar amount of time and it is about 5% of the simulation time. The longer the simulation runs, the closer the total amount of times of each peer node spent as a replica.

3.4.4 Average Network Latency Between Replicas and Client Peer Nodes

In our simulation, the peer nodes generate requests following the exponential distribution with $\lambda = 0.001$.

Figure 7 compares the average latency between every peer node and its closest replicas when different number of replicas are placed through different replica placement algorithms. The x axis represents the number of replicas deployed. The y axis represents the average latency between client peer nodes and their closest replicas. This results show whether replicas are placed in the vicinity of majority of requests. We compare the performance

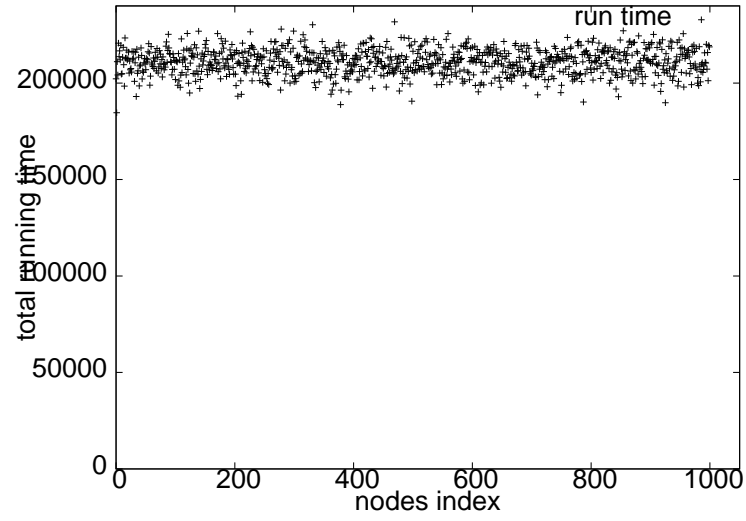


Figure 6: Total amount of time that each node contributed to replicate the service

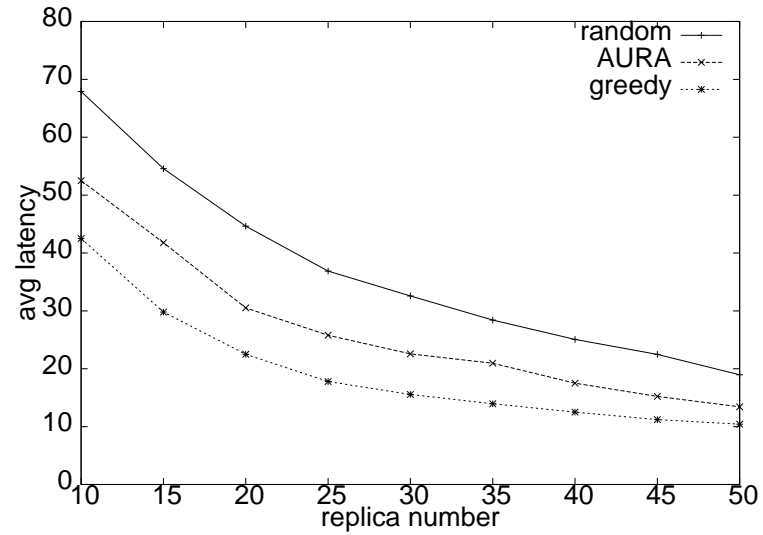


Figure 7: Average latency between peer nodes and their nearest replicas when different fraction of replicas are created

of AURA with greedy algorithm [84] and random algorithm. Figure 7 shows that AURA algorithm performs worse than greedy algorithm, but much better than the random algorithm. The greedy algorithm is a centralized algorithm and its computational complexity is $O(kN^2)$, where k is the number of replicas and N is the total number of nodes. However, it performs within a factor of 1.5 times of the super-optimal algorithm in the median cases and around a factor of 4 in the maximum case [84]. The greedy algorithm does not consider fairness. Therefore, it computes one set of replicas based on the perfect knowledge of the underlying topologies and the rate of requests generated from each node, and then use the m replicas unless the topologies or requests rates change. The random algorithm selects k nodes randomly and uses them as the replicas. We use the average of the results of 40 runs for random algorithm. To be comparable with algorithms that are not fairness aware and load adaptive, we use a fixed topology and keep the request rates of all peer nodes unchanged in this experiment. For our algorithm, we take snapshots of the replica placements generated with AURA periodically, and use the average of the results of 40 snapshots.

3.4.5 Experiments with PlanetLab

Since the simulation does not reflect the dynamics of the network latency, the overhead of query and heartbeat, and the shared load of CPU resources of the peer nodes, we measured several features of our replication algorithm on top of the PlanetLab test bed.

We randomly choose 100 nodes that are mainly distributed around north America, Europe, and a few in Asia. The maximum round trip latency between two peer nodes is about $280ms$. Each peer node can autonomously decide whether it will create a replica of the service based on the local knowledge it collects from heartbeat messages. Each peer node generates Java RMI requests following exponential distribution of $\lambda = 0.001$. Each request need $50ms$ to be served by a replicating peer nodes. When the size of the system is small, e.g., with 100 nodes, it is hard to accurately maintain 5% replicas, i.e., 5 replicas. So in the experiment conducted on the top of Planetlab, we raised our target to be 10% and try to maintain an average of 10 replicating nodes in the system at the same time.

Figure 8 shows that the total number of peer nodes that are replicating the service at the

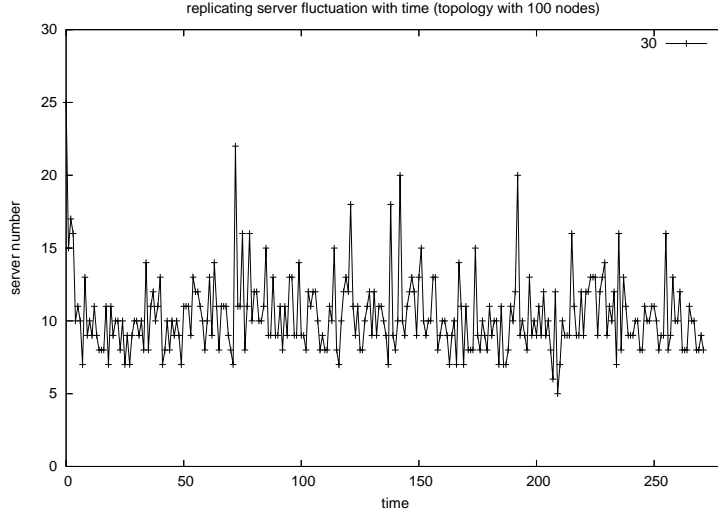


Figure 8: Variation of the number replicas in the system at the same time

same time varies around the target replica percentage of 10%, i.e., 10 nodes. The number of servers is sampled every 10000ms. The x axis represents the times that the number of servers has been sampled. The y axis represents the sample result of the replicating nodes.

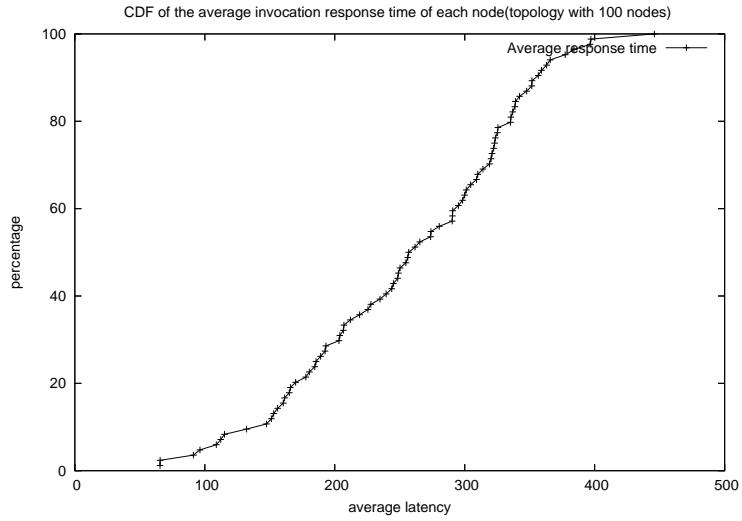


Figure 9: CDF of the average RMI call response time of all the peer nodes

Figure 9 is the CDF of the average response time over all the generated requests by each peer node. The x axis represents the average RMI call response time. The y axis represents the percentage of the peer nodes whose average response time is below certain value. We

find that the main reason why some peer nodes have much larger average response time than the majority is because there are some peer nodes we choose that are having very high load caused by other shared applications on the same peer nodes and therefore, large number of *system overload exception* on those peer nodes.

3.5 Summary

A good scheme for deciding when and where an object should be replicated is important for better utilization of resources available at peer nodes and for improving the performance of object invocation. In a peer-to-peer system, such a decision cannot be made at a central place. Instead, a distributed algorithm is needed so that each peer node can autonomously decide whether to contribute its own resources to others or not. In this section, we propose a distributed algorithm for each peer node to decide whether it needs to replicate a service locally and share it with others. The benefits of this algorithm include being responsive to heterogeneous load, ensuring that each peer node fairly contributes its available resources to meet the needs of the overall system and the ability to control consistency maintenance overhead by controlling the number of replicas in the system.

CHAPTER IV

UPDATE DISSEMINATION USING LIMITED NUMBER OF MULTICAST CHANNELS

4.1 Introduction

As discussed in previous chapters, when distributed object platforms, e.g., CORBA, DCOM and RMI are used to implement distributed applications, the latency experienced by clients can be reduced by replicating an object where it is accessed rather than always transferring its state on demand. When replicating an object, consistency must be considered between the object and its replicas. A replica of an object does not need to be totally consistent with the object. Different objects may require different levels of granularity of consistency, say, some critical objects require immediate updates to all of its replicas while other objects require only some of the updates. The replica of an object at one remote user may be more or less consistent than replicas of this object at other users. The consistency of a replica can depend on the remote user's available network resources. For example, a workstation with a broadband network connection can maintain a more consistent replica of an object than a PDA with a wireless connection.

In this chapter, we concentrate on the following problem: how updates to an object can be efficiently disseminated to its replicas to ensure that the consistency requirement are met? This problem is interesting when the number of remote users and replicated objects is very large and network resources are limited. The key to scaling applications in a heterogeneous environment with limited network resources is how the state updates to an object are disseminated to replicating users.

Two transport technologies can be used to disseminate state updates to replicating users. The first option is to use unicast, where the state update to an object is transmitted to

only one remote replicating user at a time. However in a distributed interactive application, multiple remote users can replicate overlapping sets of objects. As a result, unicast will use the object's host's outgoing bandwidth inefficiently. Another option is to use multicast. A multicast channel can be allocated for each replicated object, and remote users who replicate this object can listen to this channel. However, multicast channels are not free: they consume resources inside the network, i.e., router memory for storing multicast information[37]. The number of multicast channels available for use by an application is limited. Several objects must be grouped together into the same multicast channel. However a new problem arises: a remote user can receive updates for objects that it does not replicate but are grouped with objects it does replicate. The result is that the remote user's incoming bandwidth can be used inefficiently. The key to making large-scale distributed network applications feasible is to group objects into multicast channels in an intelligent manner so that the following requirements are satisfied:

1. Object server sends data that does not exceed its available bandwidth;
2. Replicating user receives data that does not exceed its available bandwidth;
3. Only limited number of multicast channels are used.

In this chapter, we propose a model for the problem of finding adequate groupings of replicated objects and replicating users into finite number of multicast channels. Our model takes into consideration users' heterogeneous resource constraints and can support replicated objects at varying degrees of consistency. We also propose a greedy-heuristic algorithm to solve this problem. Our algorithm is *incremental*: as the set of objects that remote users replicate changes, a new grouping is derived from the previous grouping. This allows the number of expensive join and leave operations to be minimized and the running time to be low. Our algorithm is also *adaptive*: if an adequate grouping cannot be found in a timely manner, the problem can be made easier as allowed by the policy of the application. For example, objects replicated at low-degree of consistency at a remote user could be dropped in favor of preserving the consistency of more important objects.

4.2 *Group Arrangement Problem*

4.2.1 Example: Massive Multiplayer Game

As an example, consider an MMG with thousands of players interacting in a sports event in a large stadium. Players, or more concretely, the avatars of players, can have different roles in this sports event: they can be participating directly in the sports, or they can merely be spectators in the bleachers. Players of the MMG have interest in various game entities in the stadium according to their role in the game. Players may be highly interested in other players in the field, and they might be lightly interested in score board. Spectators can be interested in the players in the field, the scoreboard, the announcer, and other spectators.

When such an application is implemented over a distributed object platform, game entities could be designed as a set of objects, e.g. locations, outlines, colors and textures, which incrementally define more details about the entity. For displaying an entity with different levels of details, different amount of objects are required. Objects could be implemented at a game server and replicated at client hosts. Updates to an object are disseminated to clients that are replicating this object. The consistency of objects and their replicas determines whether a smooth motion for objects is provided.

4.2.2 Layered Preferences and Adaptive Grouping

We focus on the update dissemination from a server to its clients. Each client has a set of replicated objects and consistency settings on these objects. These objects are updated at the server node.

Given a certain amount of system resources, i.e., number of multicast channels, server's outgoing bandwidth and clients' incoming bandwidth, the possibility of finding a feasible group arrangement depends on both system resources and client preferences. The more updates the clients ask for and the lower resources the system has, the lower the possibility of finding such a grouping

To increase the possibility of finding a feasible group arrangement, the clients can choose to be very conservative. They can ask for minimum amount of data — less replicated objects and less frequent updates. This often means lower level of game image quality and

meanwhile the system resources are not fully exploited. On the other hand, the clients can choose to ask for a lot of data to achieve higher game quality and resource utilization but risk the chance of not finding a feasible group arrangement. Because each client acts independently in such a distributed system, it is difficult for the clients to give “suitable” preferences that neither overload nor waste the system resources. To solve this problem, we propose *layered preference and adaptive group arrangement* strategy as follows: When clients submit preferences, they assign different priorities to different objects; the server starts to search a feasible group arrangement that meets all the preferences without concerning the priorities. If it can not succeed, it drops preference on some or all low priority objects and tries again. This step is repeated until a feasible group arrangement is finally found.

In the rest of this chapter, we only focus on the single step in the above iterative procedure. That is, when we try to find a feasible group arrangement, we are concerned only with the preferences and ignore the associated priorities.

4.2.3 Examples of Group Arrangement

We give several simple examples of group arrangement here. We use *source* to substitute for *object* that the object server is sending updates for, *receiver* for a *client* that is replicating some objects implemented by a server, and *channel* for a *multicast channel*. Source, receiver and channel are denoted by s_i , r_i and c_i .

Table 2 shows the outgoing bandwidth of server, incoming bandwidth capacity of seven receivers and their preferences to seven sources.

We try several possible group arrangements and show how they may not meet all the requirements.

- Case 1: Table 3 shows a grouping where all sources are partitioned into three channels. But r_3 's incoming bandwidth is overloaded.
- Case 2: Table 4 shows a group arrangement where receivers are partitioned into three channels. Server's outgoing bandwidth is exceeded by four; r_3, r_4, r_5 's incoming bandwidth is overloaded.

Table 2: Requirement Conditions

receiver	receiver's preference	B/W
r_1	(s_1, s_2, s_3)	4
r_2	(s_1, s_3, s_4)	4
r_3	(s_1, s_4, s_5)	4
r_4	(s_1, s_5, s_6)	4
r_5	(s_1, s_5, s_7)	4
r_6	$(s_1, s_2, s_3, s_6, s_7)$	8
r_7	$(s_1, s_2, s_3, s_4, s_5)$	8
Server		12

- Case 3: Table 5 shows a **feasible** group arrangement.

Table 3: Case 1

channel	receivers	sources
c_1	$r_1, r_2, r_3, r_4, r_5, r_6, r_7$	s_1
c_2	r_1, r_2, r_3, r_6, r_7	s_2, s_3, s_4
c_3	r_3, r_4, r_5, r_6, r_7	s_5, s_6, s_7

Table 4: Case 2

channel	receivers	sources
c_1	r_1, r_2	s_1, s_2, s_3, s_4
c_2	r_3, r_4, r_5	s_1, s_4, s_5, s_6, s_7
c_3	r_6, r_7	$s_1, s_2, s_3, s_4, s_5, s_6, s_7$

Table 5: Case 3

channel	receivers	sources
c_1	r_1, r_2, r_6, r_7	s_1, s_2, s_3, s_4
c_2	r_3, r_7	s_1, s_4, s_5
c_3	r_4, r_4, r_6	s_1, s_5, s_6, s_7

4.2.4 Problem Definition

We represent receivers' diverse preferences with an $m \times n$ matrix, named *preference matrix* and denoted as M^{rs} . m and n are the numbers of receivers and sources, respectively. If

receiver r_i prefers source s_j , $M_{i,j}^{\text{rs}}$ equals to 1, otherwise it equals to 0. Put in another way, row i describes the sources that r_i prefers and column j describes the receivers that prefer s_j . Figure 10(a) shows the preference matrix for example in Table 2.

	s1	s2	s3	s4	s5	s6	s7		c1	c2	c3
r1	1	1	1	0	0	0	0	r1	1	0	0
r2	1	0	1	1	0	0	0	r2	1	0	0
r3	1	0	0	1	1	0	0	r3	0	1	0
r4	1	0	0	0	1	1	0	r4	0	0	1
r5	1	0	0	0	1	0	1	r5	0	0	1
r6	1	1	1	0	0	1	1	r6	1	0	1
r7	1	1	1	1	1	0	0	r7	1	1	0
(a) M^{rs}								(b) M^{rc}			

	s1	s2	s3	s4	s5	s6	s7		s1	s2	s3	s4	s5	s6	s7
c1	1	1	1	1	0	0	0	r1	1	1	1	1	0	0	0
c2	1	0	0	1	1	0	0	r2	1	1	1	1	0	0	0
c3	1	0	0	0	1	1	1	r3	1	0	0	1	1	0	0
								r4	1	0	0	0	1	1	1
								r5	1	0	0	0	1	1	1
								r6	2	1	1	1	1	1	1
								r7	2	1	1	2	1	0	0
(c) M^{cs}								(d) $A^{\text{rs}} = M^{\text{rc}} \times M^{\text{cs}}$							

Figure 10: Example of Preference and Subscription Matrices

The system resources are defined by a server outgoing bandwidth B^{S} , number of channels k and clients' incoming bandwidth vector B^{r} . For example, in Table 2, $k = 3$, $B^{\text{S}} = [12]$, and $B^{\text{r}} = [4, 4, 4, 4, 4, 8, 8]$.

When grouping receivers and sources into channels, we use another two 0-1 matrices to represent the group arrangement: a *receiver subscription matrix* and a *source subscription matrix*. The former, denoted by M^{rc} , is an $m \times k$ matrix that describes how receivers are related to channels; $M_{i,k}^{\text{rc}}$ equals to 1 means receiver r_i listens to channel c_k . The latter, denoted by M^{cs} , is a $k \times n$ matrix that describes how sources are related to channels; $M_{k,j}^{\text{cs}}$ equals to 1 means source s_j sends updates to channel c_k . The two subscription matrices corresponding to the example in Table 5 are given in Figure 10(b) and Figure 10(c).

The product of M^{rc} and M^{cs} , denoted by A^{rs} , represents the actual amount of data received by receivers. Entry $A_{i,j}^{\text{rs}}$ is the actual number of times that r_i receives s_j . Figure 10(d) shows the value of A^{rs} corresponding to Table 5.

A feasible group arrangement must satisfies the following conditions:

Preference Satisfaction Receivers' preferences must be met.

$$\forall i, \forall j, A_{i,j}^{\text{rs}} \geq M_{i,j}^{\text{rs}} \quad (5)$$

Bandwidth Limitation Server and receivers' traffic must not exceed their bandwidth.

$$\sum_{t=0}^k \sum_{j=0}^n M_{k,n}^{\text{cs}} \leq B^s \quad (6)$$

$$\forall i, \sum_{j=0}^n A_{i,j}^{\text{rs}} \leq B_i^r \quad (7)$$

4.3 A Heuristic Searching Algorithm

4.3.1 General Idea

As previously discussed, a feasible group arrangement must satisfy four conditions. In practice, the server usually has much larger bandwidth than the receivers does. If a group arrangement satisfies condition (1) and (3), it is very unlikely that the arrangement does not satisfy condition (2). Therefore, we do not explicitly attack the satisfaction condition on the server traffic in our algorithm, but try to reduce the server traffic when possible.

Our algorithm starts from an initial group arrangement that uses the given number of channels and meets the receiver preferences. First, we must check if the initial group arrangement satisfies receiver bandwidth conditions. If it does, it is a feasible group arrangement and the algorithm halts. Otherwise it must be adjusted into another group arrangement that still satisfies receivers preferences and uses k channels. The adjustment is repeated until it satisfies the receivers' bandwidth condition. The pseudo code for the algorithm is shown in Figure 11. We will go into detail about the single adjustment step in the next subsection, but first we present two practical issues about our algorithm.

First, if network resources are very low relative to receiver preference requirements, a feasible group arrangement either may not exist, or it might take a very long time to find. There is no formal way to find out. Given the real time nature of our system, we limit the searching time of our algorithm. If a feasible group arrangement cannot be

p - Preference matrix, g - Grouping, c - Channel, l - Connection

```

group( $p$ )
 $g$  = initial_grouping( $p$ );
while (overloaded_receivers( $g$ ))
     $g$  = merge( $p$ , split( $p$ ,  $g$ ));
return  $g$ ;

split( $p$ ,  $g$ )
while (overloaded_receivers( $g$ ))
     $r$  = most_overloaded( $g$ );
     $c_0$  = sends_most_waste( $g$ ,  $r$ );
    ( $c_1$ ,  $c_2$ ) = partition( $p$ ,  $c_0$ );
     $g$  =  $g - c_0 + c_1 + c_2$ ;
return  $g$ ;

merge( $p$ ,  $g$ )
while (too_many_channels_used( $g$ ))
    ( $c_1$ ,  $c_2$ ) = least_bad_waste( $p$ ,  $g$ );
     $c_3$  = merge( $c_1$ ,  $c_2$ );
     $g$  =  $g - c_1 - c_2 + c_3$ ;
return  $g$ ;

partition( $p$ ,  $c$ )
 $c_0$ ,  $c_1$  = empty_channel;
while (!empty( $c$ ))
    ( $l$ ,  $c_2$ ) = most_reduced_waste( $p$ ,  $c$ ,  $c_0$ ,  $c_1$ );
     $c$  =  $c - l$ ;  $c_2$  =  $c_2 + l$ ; //  $c_2$  is either  $c_0$  or  $c_1$ 
return ( $c_0$ ,  $c_1$ );

```

Figure 11: Split/Merge Greedy Heuristic Algorithm

found within the specified time limit, the algorithm is stopped and the receiver preference requirements are relaxed in an application specific manner, then the algorithm is run on this new problem. This allows the algorithm to adapt to low resource conditions or aggressive receiver preferences. On the other hand, receiver preference requirements can be changed in an application specific manner to fully utilize available network resources.

Second, there are several ways to generate an initial group arrangement. However it is better to start from the previously feasible group arrangement than a randomly generated group arrangement and *improve* it to satisfy the new receiver preference. By using the previous feasible group arrangement, new group arrangement can be derived from modifying

the previous feasible group arrangement gradually. So the algorithm is more likely to find the new feasible group arrangement more quickly than if it started from a random initial group arrangement. This also has the added benefit of reducing the amount of costly multicast join and leave operations. In this sense, our algorithm is **incremental**.

4.3.2 Two Phases of A Single Adjustment Step: Splitting and Merging

First, we explain some terminology that is used:

- *waste* is the unwanted data received by a receiver;
- *bad-waste* is the waste that exceeds a receiver's bandwidth capacity. It is the amount of data received minus receiver's bandwidth capacity if a receiver receives more data than its bandwidth capacity; otherwise is 0.
- *distance* between connection $L_1 [r_i, s_j]$ and connection $L_2 [r_x, s_y]$ is called $D(L_1, L_2)$; $[r_i, s_j]$ is a *connection* if r_i is interested in s_j . Distance between two connections (or sources or receivers) describes the similarity between two connections/sources/receivers.

$$D(L_1, L_2) = w_1 \times D(r_i, r_x) + w_2 \times D(s_j, s_y)$$

w_1 and w_2 are weight of $D(r_i, r_x)$ and $D(s_j, s_y)$ correspondingly

$$D(r_i, r_x) = \frac{|sources(r_i) \cup sources(r_x)|}{|sources(r_i) \cap sources(r_x)|}$$

$sources(r_i)$ represents the set of sources that are preferred by r_i ; similarly, $D(s_j, s_y)$ can be defined.

$$D(s_j, s_y) = \frac{|receivers(s_j) \cup receivers(s_y)|}{|receivers(s_j) \cap receivers(s_y)|}$$

$receivers(s_j)$ represents the set of receivers that are interested in s_j . If the overlap between two sets is empty, the size of the overlap set is set to be 1.

When any receiver is overloaded, we use the *split* operation. First, the most overloaded receiver is chosen. Then, the channel that brings the most *waste* to the overloaded receiver is chosen. This chosen channel is split into two sub channels in the way that reduces the **most bad-waste**. For a channel with x receivers and y sources, there are 2^{x+y} ways of splitting it into two sub channels. This is computationally too expensive and unnecessary.

In our algorithm, splitting a channel into two subchannels to reduce receivers waste is done by partitioning the *connections* $[r_i, s_j]$ in the channel into two groups. *Partition* here means that the partitioned entities will appear in only one group, not both. The six solid lines in Fig 12(a), representing six connections, are partitioned into two groups in Fig 12(b). From (a) to (b), dashed lines, representing the waste data received, are reduced by two. We want to reduce the **most** waste data for receivers when splitting a channel. The code for connection partition is shown in Fig 11. By splitting the connections in a channel, we are not biased in favor of either receivers or sources. This frees us from doing just receiver partitions where a receiver can only appear in one channel or source partitions where a source can appear only in one channel [110]. In our approach receivers and sources will appear in any number of channels as long as it reduces more receivers bad waste. The splitting operation is repeated until no receiver is receiving bad waste anymore. However, it is possible that we may be using more than k channels.

When more than k channels are used, we will use the *merge* operation to combine two channels into one. Merging c_1 and c_2 in figure 12(b), we can get c_1 in Fig 12 (a). After the merge, the server needs to send out only three source objects, instead of five. However, receivers r_2 and r_3 now receive more waste. In general, merging two channels will cause receivers to receive more waste and the server to send less data. The increased bad-waste (or waste) at receiver side after merging two channels into one is calculated for each pair of channels. We choose the pair of channels increases the *least bad-waste* (or *waste* if *bad-waste* ties) at receiver side. The merge operation is repeated until only k channels are left.

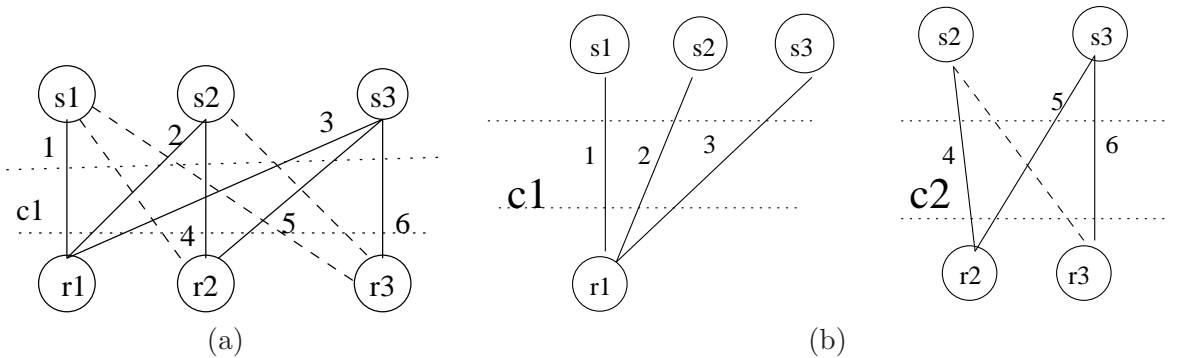


Figure 12: Connections Partition

4.3.3 Computation Complexity

When splitting a channel with m receivers, n sources and c connections, c is bounded by $m \times n$, into two subchannels, all c connections in the original channels will be moved to either of the two new subchannels one by one. The overhead of moving one connection is the overhead of calculating the reduced bad waste of moving one connection to the new subchannels for each remaining connection in the original channel. So the complexity of splitting one channel into two is $O(c^2)$. When merging x channels into $x - 1$ channels, we need to choose the two channels that reduce the most bad waste (or waste) from the x channels. The overhead of choosing two channels from the x channels is x^2 .

If x is the max number of channels used when no receiver is overloaded, the computation complexity of one loop $O(x^3 + xc^2)$. The number of loop iterations is bounded by an input parameter, which is used to reduce running time to reasonable limits for a server.

4.4 *Evaluation*

We conduct some simulation experiments and show that our greedy grouping is better than some other existing grouping algorithm.

4.4.1 Generating Preference matrix

We generate preference matrices with a simulated multiplayer game. We use a model similar to that used in [121]. The virtual world of the multiplayer game is a rectangular battlefield. There are two separate sets of entities in the battlefield: receivers and sources.

To make our algorithm comparable with cell-based algorithm, vision domain circles around receivers are used for calculating preference matrices. To simulate diverse receivers' preferences and sources' popularity, we assign random brightness to each source and assign random vision domain size to each receiver. We define receivers' preferences as follows: if the distance between a source and a receiver is less than the receiver's vision domain size multiplied by the brightness of the source, then this source is in this receiver's preference set.

By configuring sources with different brightness and receivers with different vision domain, we can simulate diverse preferences.

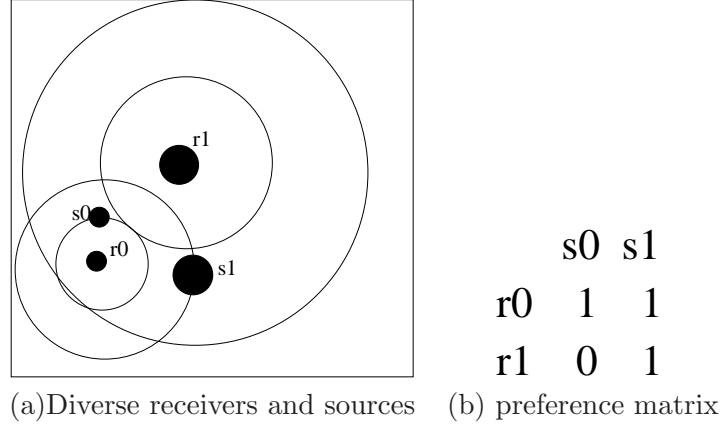


Figure 13: Diverse Receivers and Sources

In Fig 13 (a), brighter source $s1$ can be seen by a receiver if it is within its bigger circle, less bright source $s0$ can be seen if within its smaller circle. Its corresponding preference matrix is Fig 13 (b).

Receiver and sources move around the virtual world at varying velocities. During a period of time, we can get a sequence of preference matrices, each of which is created by collecting all receivers' preferences at a given moment. This sequence of matrices can be used as the input for different grouping algorithms.

We also simulate network resource conditions. We set values for the number of available multicast channels, receiver incoming bandwidth, and server outgoing bandwidth. We vary these values to see performance of grouping algorithms under different resource conditions.

Finally, we want to bound the running time of our algorithm because of the possibility of not finding a group arrangement. In our experiment, we choose to reduce the game quality by ten percent if no group arrangement can be found within the time bound. Reducing game quality by ten percent means ten percent less data are used for updating the same object, so ten percent less network bandwidth is needed. We will keep reducing ten percent of the game quality until a group arrangement is found. The game quality supported by cell-based grouping can be obtained in a similar way although no running time bound is

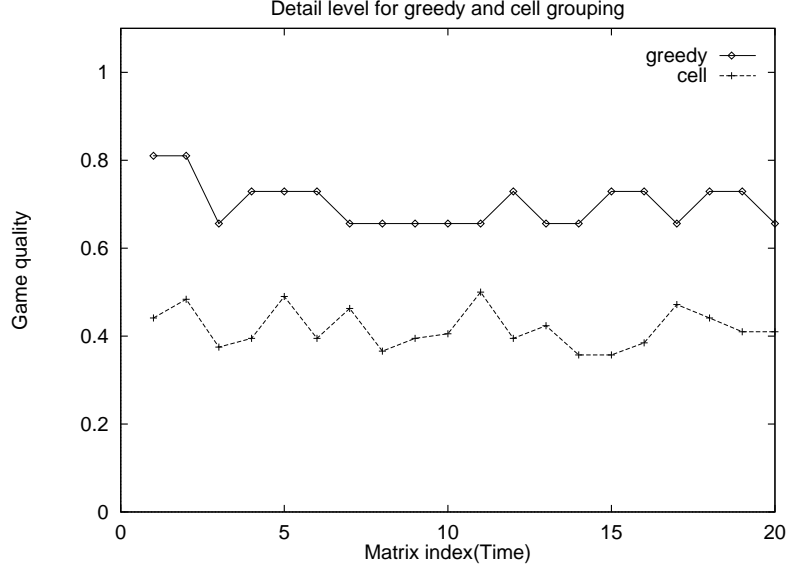


Figure 14: Comparison with Cell-based Grouping

needed and no gradually decreasing game quality is needed. The final game quality is used to compare the effectiveness of different grouping algorithm.

4.4.2 Comparison With Cell-based Algorithm

In cell-based algorithm, the battlefield is evenly divided into k rectangle cells, each of which is associated with a multicast channel. The receivers listen to the multicast channels associated with the cells that overlap with their vision domain.

For each sequence of preference matrices, we compare the game quality supported by different algorithms. This simulation has 100 receivers and 100 sources. 30 receivers have vision circle radius of 40, bandwidth capacity of 15; 30 receivers have vision circle radius of 60 and bandwidth capacity of 25; 40 receivers have vision circle radius of 80 and bandwidth capacity of 35. Fig 14 shows our algorithm supports higher game quality than cell-based algorithm.

Second we show the trend of game quality when resource conditions change. Fig 15 shows the different game qualities supported when 4, 9, 16, 25 multicast channels used. 50p means 50 receivers; 60v means vision domain size of 60; 4c means available channels are 4. The figure shows that as more channels are used, the quality supported becomes better.

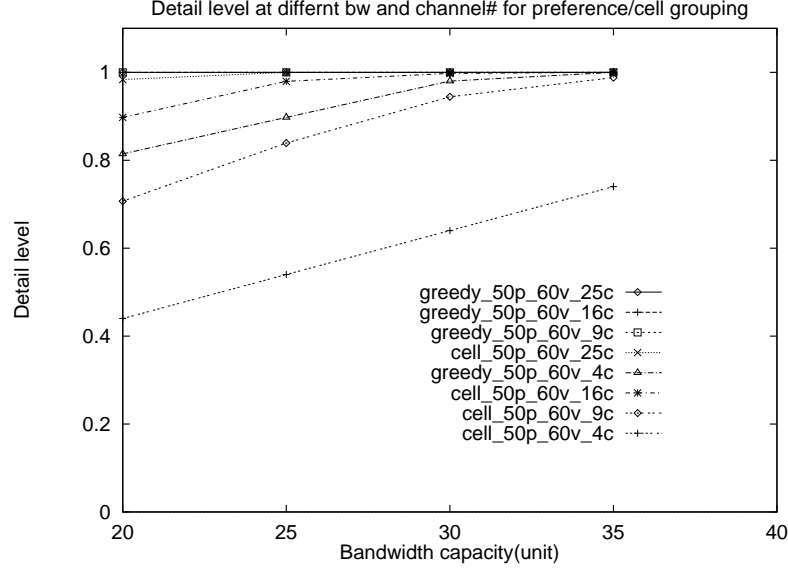


Figure 15: Comparison with Cell-based Grouping

Too few channels can provide only very low game quality; but too many channels are not necessary either.

With cell-based algorithm the receivers simply receive data from all the cells that overlap with their vision domains. When grouping, it does not consider individual client's specific preference and bandwidth constrain. How much data a receiver can get depends only on the location of the receiver in the battlefield and not on its actual bandwidth. So receivers can receive large amount of waste data and the overall game quality is degraded. When same number of multicast channels are used, our greedy algorithm is always able to support higher game quality. This suggests our algorithm provides higher game quality under same level of network resource conditions. We see a similar trend when we run the experiment under several other resource configurations.

4.4.3 Joins and Leaves During Regrouping

When receivers' preferences change, group arrangement changes. Since joins and leaves of members can introduce overhead to the maintenance of a multicast channels, a better grouping algorithm should incur less joins and leaves during regrouping.

Fig 16 shows that our group arrangement incurs much less joins during regrouping given

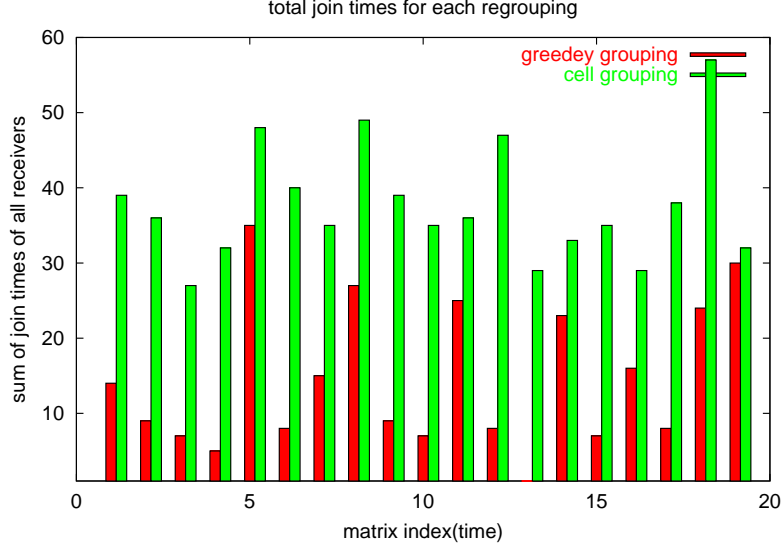


Figure 16: Comparison with Cell-based Grouping

same resource conditions and same sequence of preference matrices, since our grouping is incremental. The result for leaves is similar.

4.4.4 Connection Similarity Vs. Receiver(source) Similarity

Receiver(source)-based clustering is proposed in [110]. Because [110] intends to reduce the **overall** traffic received by all receivers rather than finding a feasible group arrangement that satisfies given resource conditions, it is hard to compare its performance with that of ours. But it would be interesting to make a comparison between the two by having a closer look into their underlying methodologies.

We focus on the task of channel splitting, which is common in both algorithm. In Figure 11, this task is carried by the code segment **partition**, which make use of the similarity among connections. In [110], it is done by clustering receivers(sources) based on the similarity among receivers(sources). It has been shown in [110] that receiver-based clustering is suitable for multiplayer gaming and source-based clustering is suitable for stock quote streaming. However, it is possible that an application does not have a constant preference pattern.

To highlight the difference between connection and receiver(source) similarity, we use

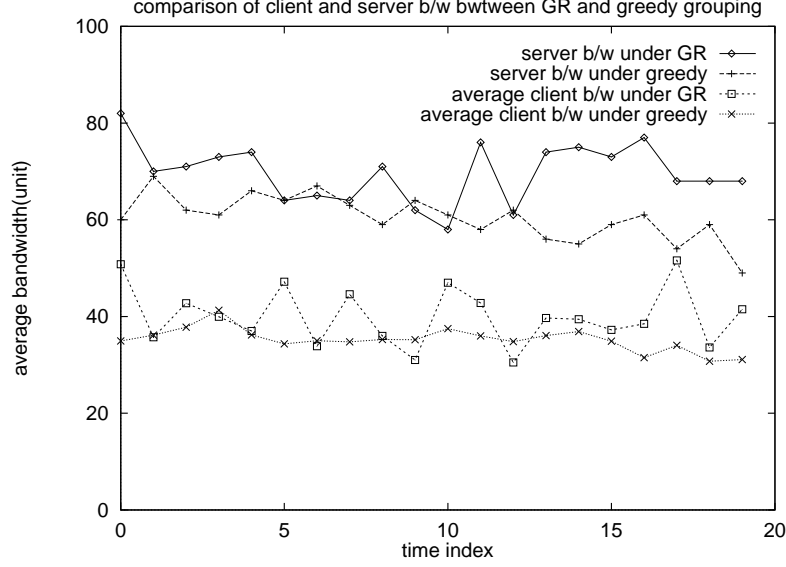


Figure 17: Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR)

the k -means clustering package in MATLAB to split a channel based on receiver-similarity and compare the result with that of our algorithm. We examine three cases. In first case, all sources have same brightness and all receivers have same vision domain size and bandwidth capacity. As shown in Fig 17, two algorithm result in similar amount of traffic at receivers and server. In second case, all sources have same brightness but there is large variation in receivers' vision domain size and bandwidth capacity. As shown in Fig 18, the two algorithms result in similar amount of incoming traffic at the receivers but less outgoing traffic at the server. In the last case, there are large variance in sources' brightness and receivers' vision domain size and bandwidth capacity. As shown in Fig 19, our algorithm results in less receiver incoming traffic and less server outgoing traffic.

4.5 Summary

We have presented an incremental and adaptive heuristic-based algorithm that can group sources and receivers in a way that it handles limited network resources. We have shown through experiments that our algorithm can produce better results than several algorithms that have been developed in the past for update dissemination.

Since our algorithm considers individual receivers' bandwidth capacity, it can migrate



Figure 18: Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR)

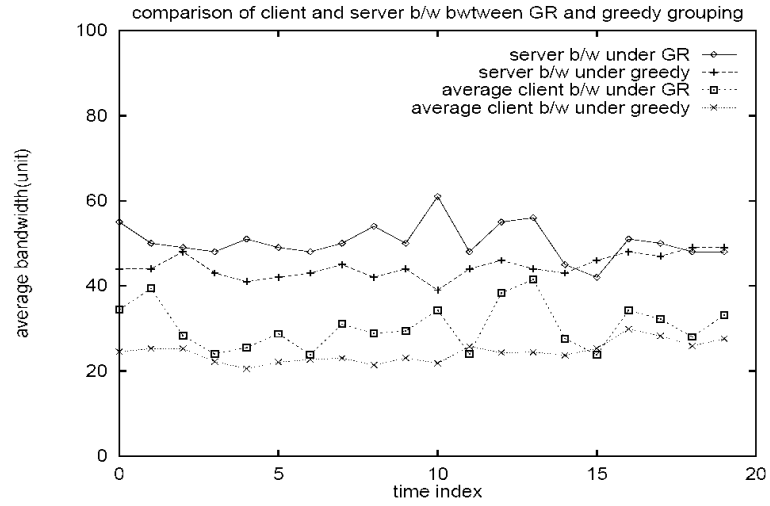


Figure 19: Using Connection Similarity(greedy) Vs. Using Receiver Similarity(GR)

bad-waste from overloaded receivers to the receivers that are not overloaded, thereby it better preserves the bandwidth of low-end users. Also, since a new feasible group arrangement is incrementally derived from a previous feasible one, number of joins and leaves of multi-cast channels is reduced. In addition, our algorithm allows both receivers and sources to subscribe to multiple channels and therefore are suitable for a wide range of applications.

CHAPTER V

UPDATE DISSEMINATION WITH OVERLAY NETWORK

Peer-to-peer networks and mobile ad hoc networks are emerging distributed networks that share several similarities. Fundamental among these similarities is the decentralized role of each participating node to route messages on behalf of other nodes, and thereby, collectively realizing communication between any pair of nodes. Messages are routed on a topology graph that is determined by the peer relationship between nodes. Although routing is fairly straightforward when the topology graph is static, dynamic variations in the peer relationship that often occur in peer-to-peer and mobile ad hoc networks present challenges to routing.

5.1 *Introduction*

Group communication requires efficient delivery of the same content to multiple recipients. Due to the difficulty in deploying IP multicast at the Internet scale, the use of overlay networks has been proposed as an alternative solution to support group communication (e.g., [31, 18, 81, 71]). In an overlay network, overlay nodes cooperate with each other to forward data on behalf of any pair of communicating nodes in the application; each node not only generates and sinks but also forwards traffic. Although in many proposals overlay networks are constructed over volunteering end-systems, researchers have recently paid increasing attention to infrastructural overlay networks [83, 101] composed of hundreds of dedicated nodes. These dedicated nodes are typically deployed by Overlay Service Providers(OSPs), either at the edge or in the core of the Internet, to provide generic support to a variety of applications [70], including those based on group models such as collaborative environments, distributed computation, online conference and messaging, and massive multiplayer games. Sharing the same pool of dedicated overlay nodes, each application can construct its own

overlay network and tailor it for its own purpose. Positioned between the underlying IP networks and the application running on top, the overlay network can be configured with monitored application-level communication characteristics in mind and be optimized based on application specific performance metrics. One important configurable component of an overlay network is its topology. Although some overlay networks use completely meshed topology (e.g., [12]), a degree bound is often necessary and favorable for large scale overlay networks due to various concerns in scalability [70].

In this chapter, we study how the heterogeneity in users' preferences to data affects the configuration of an overlay network, particular, the choice of its *overlay topology*, in group based applications. In many of these group based applications, users' *preferences* to data could be very diverse: the application data could belong to many topics and each user could be interested in one or many topics; every user interested in a topic is both a generator and a receiver of the data on that topic. When the application has many topics of data and each node has heterogeneous preference for those topics, due to the cooperative nature of overlay, some nodes may be forwarding data they do not prefer, termed *waste* in this chapter, for other nodes and this incurs more overlay traffic. This is shown in an intentionally simplified example as follows.

There are four overlay nodes (A, B, C and D) and four topics (W, X, Y and Z) in the example. Figure 20 (a) shows the network distance between every pair of nodes and each node's preference of topics.¹ Assuming a degree bound of 2, Figure 20 (b) and (c) show two example overlay topologies, and (d) and (e) show the flow of data over the two topologies, respectively, assuming shortest path routing in the overlay level. Topology I sets up overlay links only between nodes with overlapping preferences of topics. It incurs no waste. Topology II sets up an overlay link between nodes A and D (similarly, between B and C) although they have totally different preferences. The short network distance between A and D allows for shorter overlay paths, at the price of increased waste at D . Table 6 compares

¹In practice, subscription happens between end users and the topics. However, when end users register their preferences at their dedicated *entrance* overlay nodes, the subscription turns into a relationship between the overlay nodes and the topics. To simplify the treatment, here and in the rest of the paper we ignore the presence of end users and focus on the overlay nodes. This does not affect the essential complexity of the problem and the approach we use.

the overall latency and overall waste incurred in the two different topologies, assuming each node generates one unit of data for each of the topics it prefers.

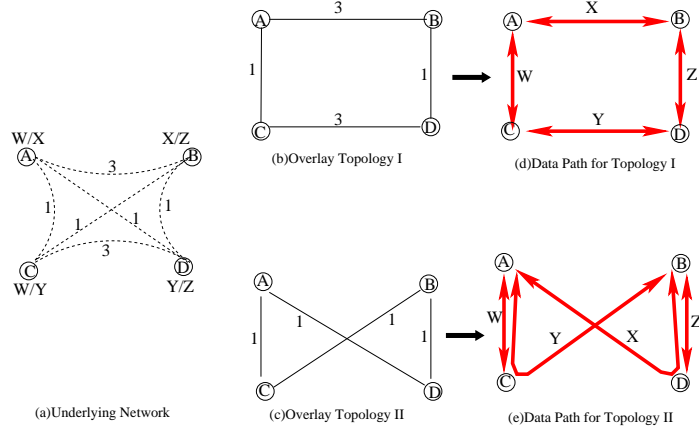


Figure 20: Example of Different Overlay Topologies and Different Data Path

	Topology I	Topology II
overlay latency between A and B	3	2
overlay latency between A and C	1	1
overlay latency between B and D	1	1
overlay latency between C and D	3	2
overall latency	16	12
overall waste	0	4

Table 6: Different Latency and Waste for Different Overlay Topologies

End-to-end latency is an important concern in overlay design and researchers have shown that the latency can be reduced by considering network proximity when building overlay topologies [105, 89]. However, we argue that the waste is also an important concern because it directly affects the total amount of traffic incurred in the overlay network, which includes the amount of waste (topology-dependent) and the amount of data actually preferred by the nodes (topology-independent). The control of the amount of traffic is especially important in an infrastructural overlay network where overlay resources (e.g., bandwidth) are shared among many applications and a fee may be charged based on the consumption of the resources. Ideally, an overlay topology should minimize both the latency and the waste. The previous example shows the conflict between the latency property and the waste property of overlay topologies — when the degree of an overlay topology is bounded, the overlay nodes are in a dilemma whether to establish overlay links to nodes in vicinity (which helps to

reduce latency) or to those with similar preference of topics (which helps to reduce waste).

In this paper we study overlay construction methods that allow the applications to build overlay topologies with both the end-to-end latency and the amount of waste data in mind. To the best of our knowledge, there are no solutions taking into account both the topology awareness and the content awareness in the overlay network construction. We present an approach that uses multi-attribute clustering of overlay nodes to facilitate the construction of overlay topologies. The approach considers both nodes' preferences and their network locations during the multi-attribute clustering to make tradeoffs between the end-to-end latency and the amount of waste data in the resulting overlays. The approach is tunable, allowing the applications to pursue different tradeoff points as they desire.

5.2 Problem Formalization

In this section, we introduce the notations, state our assumptions and formally define the problem of overlay topology construction in the presence of heterogeneous user preferences.

First, we assume that the application data has been categorized into a set of topics, denoted with a vector $T = \{t_1, t_2, \dots, t_m\}$. We also assume there are n nodes in the system and denote them as a vector $U = \{u_1, u_2, \dots, u_n\}$. Each node could be interested in one or more topics. We represent the preferences relationship between topics and nodes with a $n \times m$ *preference matrix* P . Each element in the matrix, P_{ij} , takes a binary value 0 or 1: P_{ij} equals 1 if node u_i is interested in topic t_j and therefore might not only generate data on topic t_j but also want to receive data on topic t_j ; and P_{ij} equals 0 otherwise. Again, for simplicity of discussion, we assume that each node generates comparable amount of data on each topic that need to be delivered to every other nodes.

We use a *distance matrix* D to denote the routing latency between every pair of nodes through the *native Internet substrate*, where D_{ij} is the routing latency from node u_i to node u_j through the native network. Note that D_{ij} does not depend on whether there is overlay link between node i and j or not. We assume D is symmetric, i.e., $D_{ij} = D_{ji}$.

To deliver data from senders to receivers, an overlay network connecting all nodes needs to be built and an overlay topology needs to be chosen. The overlay topology is described

using a graph $G = (U, E)$, where U is the set of nodes and E is the set of overlay links between nodes. There are $2^{n(n-1)/2}$ possible overlay topologies, but practically we are interested in only the connected and degree-bounded ones and denote the degree bounds with a vector $F = \{f_1, f_2, \dots, f_n\}$ where f_i is the maximum number of overlay links node u_i can maintain. Even with these constraints, there are still a large number of candidate overlay topologies. To choose the best from these candidate overlay topologies, we consider two metrics, namely, the *overall latency* of an overlay topology, and its *overall waste*.

Overall Latency If there is an overlay link between u_i and u_j , the *length* of the overlay link $E(u_i, u_j)$ equals D_{ij} . Once the overlay topology is decided, we assume the data is routed in the overlay network using shortest path routing. Therefore the end-to-end routing latency (simply *latency* hereafter) between u_i and u_j through the overlay network is the sum of lengths of all overlay links along the shortest path joining u_i and u_j in the overlay topology. We use a *latency matrix* L to denote the latencies between every pair of nodes in overlay network with topology G . Note that different overlay topologies G result in different latency matrices L , so the latency matrix is a function of G . The overall latency of an overlay topology G is the sum of latency of all the data delivered in the overlay network assuming each node sends one unit of data per unit of time. Formally, the overall latency of G is:

$$latency(G) = \sum_{t_k \in T} \sum_{u_i, u_j \in U} L_{ij} \cdot P_{ik} \cdot P_{jk}$$

The term $L_{ij} \cdot P_{ik} \cdot P_{jk}$ equals L_{ij} if both node u_i and node u_j prefer topic t_k , and zero otherwise.

Overall Waste Waste is caused when the data of a topic is routed through nodes that do not prefer it. The overall waste of an overlay topology G is the total amount of waste data received by all nodes in a unit of time. Formally, the overall waste of G

$$waste(G) = \sum_{t_k \in T} \sum_{u_i, u_j \in U} \sum_{u_s \in Path(u_i, u_j)} P_{ik} \cdot P_{jk} \cdot \overline{P_{sk}}$$

where $Path(u_i, u_j)$ is the shortest path joining u_i and u_j in G . The term $P_{ik} \cdot P_{jk} \cdot \overline{P_{sk}}$ equals 1 if both u_i and u_j prefer topic t_k and meanwhile u_s is on the path joining them but does not prefer topic t_k ; the term is zero otherwise.

An application can define a parameter λ to reflect its emphasis on the latency and the waste. The more an application emphasizes waste, the larger is λ ; the more an application emphasizes latency, the smaller is λ . This results in a combined objective function f that is the weighted summation of the overall latency and the overall waste:²

$$f(G, \lambda) = \lambda \cdot \text{waste}(G) + (1 - \lambda) \cdot \text{latency}(G) \quad (8)$$

Our goal is: given λ , find the overlay topology G that can minimize the objective function $f(G, \lambda)$ shown in Eq. 8. When there is only one topic and all nodes prefer the single topic, the above problem is reduced to the problem of finding the overlay topology that minimizes the overall latency, which has already been shown to be an NP-hard problem in the literature [48] [111]. In section 5.3, we propose heuristic methods of constructing overlay topologies that can achieve the desired tradeoffs between the overlay latency and the overall waste of the overlay network.

5.3 *Constructing Preference-Aware Overlay Topologies*

Essentially, the process of constructing an overlay topology is the process of choosing overlay neighbors for each node. During this process, both latency and waste must be considered.

The latency of delivering data from a sender to a receiver is affected by the extent of the overlay topology being aligned with the underlying native network topology. It has been found that overlay topologies built with *proximity* of nodes in mind can significantly reduce the latency of the overlay network [105]. In principle, the extensive use of “long” overlay links should be avoided so that the data will not be routed back and forth many times between remote regions. Intuitively, if each node sets up overlay links only to nodes that are close in term of network distance, the paths will potentially be shorter but will include more nodes that have different preference; the resulting overlay topology will potentially have lower overall latency but a larger amount of waste.

On the other hand, the waste received by a node is caused by including this node in the paths for delivering data outside of this node’s preference. Intuitively, the overlay links

²Latency and waste have different units. So the choice of λ is usually based on the normalize values (e.g., in range $[0, 1]$) of the latency and waste.

between nodes with different preferences could potentially increase the probability of the nodes being in paths that relay waste data from a node’s perspective. If each node sets up overlay links only to other nodes that have similar preferences, the shortest paths will potentially be longer but will also contain less nodes that have different preferences; the resulting overlay topology will potentially have a smaller amount of waste but higher overall latency.

Because of the degree bound constraints, each node must wisely choose its limited number of neighbors and make a balance between network distance and preference. In addition, a node cannot simply choose its neighbors totally independent from other nodes’ choices because the shortest paths in the resulting overlay topology should be computed from all the nodes’ choices as a whole; it is this combined set of choices that makes the problem of overlay topology construction hard to solve.

Instead of attempting to construct the overlay topology in one step and deal with all the complexity in that single step at the same time, we propose to decompose the process of topology construction into two steps. In the first step, we cluster the nodes into clusters based on both the preferences and the network distance. In the second step, on top of the clustering resulted from the first step, we construct intra-cluster topologies for each cluster and inter-cluster topology across the clusters. By doing so, we decompose the complexity of the monolithic problem into two subproblems. The results from the first step provide a good guidance to the second step, and heuristically, by doing a good job in step one, the complexity of step two is significantly decreased.

5.3.1 Multi-Attribute Clustering of Nodes

The goal of node clustering is to reveal the relationship between the nodes, in term of their network proximity and preference. The process of clustering organizes the nodes into exclusive clusters in such a way that nodes within the same cluster are “close” to each other. The effectiveness of clustering heavily depends on the meaningful definition of closeness between nodes. In our context, the *closeness* between nodes is determined by two attributes: the network distance between nodes and the similarity between their preferences

of data. Heuristically, for a group of nodes, the more similar their preferences are and the shorter their network distances are, the closer they are.

First, we use the following Euclidean equation to measure the similarity of preferences between two nodes, node u_i and node u_j :

$$S_{ij} = \sqrt{\frac{\sum_{k=1}^m (P_{ik} - P_{jk})^2}{m}} \quad (9)$$

S_{ij} captures the preference similarity between two nodes. The smaller S_{ij} is, the more similar two nodes' preference are. $0 \leq S_{ij} \leq 1$.

Next, we define *virtual distance* that combines the two metrics, similarity of preferences (S_{ij}) and network distance D_{ij} with a tunable parameter ω . Formally, virtual distance between node u_i and node u_j , denoted by V_{ij} , is calculated using the following equation: ³

$$V_{ij} = \omega \cdot S_{ij} + (1 - \omega) \cdot D_{ij} \quad (10)$$

Our clustering algorithm works as follows. First, all n nodes are randomly initialized into K clusters. Then the algorithm goes through rounds of adjustment to clusters. In each round of adjustment, for each node u_i in cluster C_j , the algorithm moves u_i into cluster C_k if its average virtual distance to the set of nodes in cluster C_k is shorter than to those in C_j . The clustering algorithm terminates when all the clusters stabilize. As in all other k-means flavor clustering algorithms, the number of cluster K needs to be estimated beforehand. A traditional way to solve this problem is to start with an estimated K , and then tune the value of K to get the best result. The algorithm can be easily adapted to an incremental one. Instead of starting from a random grouping, clustering can start with a previously calculated grouping (except for the first time) to improve the efficiency of our algorithm. Intuitively, if the preferences change gradually, the new grouping should have much resemblance with the previous one.

³Practically, D_{ij} often needs to be normalized to a value in range $[0,1]$. There are several different ways to numerically do it, e.g., by dividing the network distance with the largest network distance between pairs of nodes.

5.3.2 Constructing Overlay Topology Based on Clustering of Nodes

The clustering of nodes resulting from the multi-attribute clustering process (Section 5.3.1) reveals the relationship between the nodes and provides a guideline for overlay topology construction. In this section, we describe methods of constructing an overlay topology from the clustering of nodes.

In the clustering, the relationship between two nodes is basically one of the two: they are either in the same cluster or in two different clusters. Intuitively, the fact that nodes within the same cluster are close to each other suggests that we should set up more overlay links between nodes within the same cluster than those in different clusters: 1) for the purpose of reducing the overall waste of the whole overlay topology, setting up more overlay links between nodes with similar preferences potentially reduces the probability that data is routed through nodes that do not prefer it; and 2) for the purpose of reducing the overall latency of the whole network, setting up more overlay links between nodes with low network distance potentially reduces the probability that data is routed back and forth between two far away nodes.

However, it is not feasible for a node to assign all its available overlay links only to nodes within the same cluster — nodes sharing the same preferences could still possibly be assigned to different clusters and overlay links across clusters are still necessary to ensure that the whole overlay is connected and data can reach every node in every cluster if necessary.

Based on the above observation, we propose the following method of constructing overlay topologies based on a clustering of nodes: nodes assign a portion (e.g., 90 percent) of their overlay links (subject to degree bound) to connect other nodes within the same clusters; meanwhile, they reserve a small portion (e.g., 10 percent) of overlay links for connecting different clusters together into a whole overlay. Thus, the process of overlay topology construction from a clustering of nodes is conducted in two phases, namely, the construction of *intra-cluster overlay topology* in the first phase, and the construction of *inter-cluster overlay topology* in the second phase.

Constructing Intra-Group Topology Assuming we reserve R_i overlay links in cluster

C_i for inter-cluster use⁴, we choose R_i nodes in the cluster (in our experiments shown in Section 5.4, we choose the R_i nodes with the largest degrees) as the *border nodes* of the cluster and each of them reserves one degree for inter-cluster topology. All the remaining degrees in the cluster are fully utilized for the intra-group topology.

One may suggest that the task of constructing the intra-cluster topology has the same complexity as the task of constructing the topology for the whole overlay. We argue that, however, since the nodes within each cluster are already close to each other in term of both their network distance and the similarity between their preference, the choice of inter-cluster overlay topology does not play as a significant role as for the whole overlay — our experiments in Section 5.4 shows that a randomly chosen inter-cluster topology (subject to the connectivity requirement and degree bound) works almost as well as a carefully chosen one.

Algorithm 2 shows the pseudo-code for constructing a random overlay topology that is connected and subject to degree bound. The algorithm starts from a complete graph and keeps deleting links until the degree bound is satisfied. The algorithm maintains the graph’s connectivity at each intermediate step. During the process, each link is in one of the three states: *pending*, *kept* and *deleted*. Initially, all links in the complete graph are in the state *pending*. Then in each step we randomly choose a pending link l adjoining a node that has not exhausted its available links, and check if the permanent keeping of l (together with the permanent deleting of the remaining pending links of the node until the node reaches its degree bound) breaks the connectivity of the graph. If it does, l is permanently kept (and the remaining pending links of the node are permanently deleted); otherwise, l is permanently deleted. The algorithm terminates when all links are permanently kept or deleted. This algorithm guarantee a connected topology that also satisfies degree constraints.

Constructing Inter-Group Topology The construction of the inter-group topology is similar to that of the intra-cluster topologies and the same algorithm can be used except

⁴Practically, there are issues in choosing the portion by which the total number of available overlay links is divided for inter-cluster topology and for intra-cluster topology. In the experiments shown in Section 5.4, we reserve 10% degree in each cluster for inter-cluster use and get good results. But by no means do we imply that it is the best choice.

Algorithm 2 Constructing Random Connected Topology Subject to Degree Bound

$G = (V, E)$, G is a complete graph

$S_{pending} \leftarrow E$

$S_{kept} = \phi$

initialize the $degree_{remain}(v)$ for each node to be given value

while $S_{pending} \neq \phi$ **do**

 randomly select a link $l = (v_i, v_j), l \in S_{pending}$ and (v_i or v_j has not satisfied its degree bound)

$S_{pending} = S_{pending} \setminus \{l\}$

Checkpoint:

$S_{kept} = S_{kept} \cup \{l\}$

$degree_{remain}(v_i)$ decreases by 1

$degree_{remain}(v_j)$ decreases by 1

if $degree_{remain}(v_i) \neq 0$ and $degree_{remain}(v_j) \neq 0$ **then**

 do nothing

else if $degree_{remain}(v_i) == 0$ **then**

 check whether G is still connected if we delete all pending links of v_i

if true **then**

 do nothing

else

 roll back to **Checkpoint**

end if

else

 do the same thing for v_j as for v_i

end if

end while

that 1) each cluster is treated as a giant virtual node, 2) after the algorithm terminates, a link connecting cluster C_i and C_j is actually established between two randomly chosen border nodes, one from cluster C_i and the other from cluster C_j while all degree constraints are satisfied.

5.4 Performance Evaluation

In this section, we conduct experiments to evaluate the performance of overlay topology constructing algorithm. In these experiments, we observe the quality of the overlay topologies resulting from our algorithm: if the overlay network is efficient in disseminating data in term of both the overall latency and the overall waste, it is a proof that multi-attribute clustering is beneficial in overlay topology construction and our methodology is valid. Through the experiments, we can also evaluate the scalability of our algorithm when the number of overlay nodes is large. During the discussion, we also address practical issues related to the usage of our algorithm, e.g., the choosing of the number of clusters in clustering and the optimal value of ω in Eq.10 for a given value of λ in Eq.8.

5.4.1 Experiment Setup

Our experiments are simulation-based. We use GT-ITM’s transit-stub model [115] to generate a router-level Internet topology with 1640 router-level nodes and randomly select a certain number of router-level nodes as the set of overlay nodes. We assign router-level link delays with random values ranging from 5 to 10ms for intra-transit, 10 to 20ms for transit-to-stub links and 1 to 3ms for intra-stub links. The network distance between every pair of nodes is calculated based on router-level link delays assuming shortest-path-routing in this router-level topology.

We use 400 overlay nodes (except the experiments for scale of overlay networks) and 500 topics in our experiments. Since our algorithm randomly selects overlay links when constructing overlay topologies, we run the experiment 20 times for each set of parameters and use the average of the 20 runs to produce the figures.

We experiment on two types of preference matrix P . **Preference type I** is generated through simulation of a rectangular field in virtual environment. Each overlay node has a

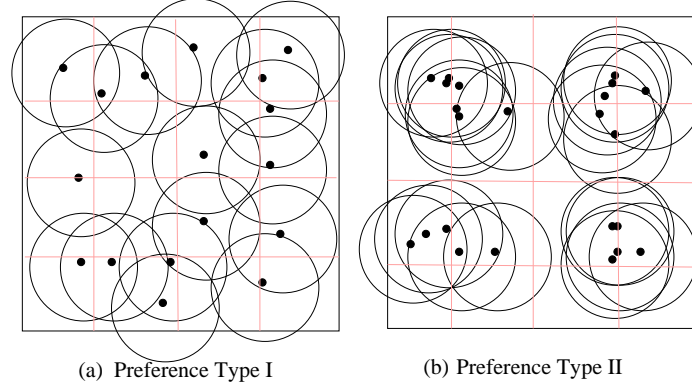


Figure 21: Generation of Preference Type I and II

virtual representation, avatar, in the virtual world and each avatar has a “vision” circle. The avatars are randomly distributed in the field. The rectangular field is divided into cells and each cell corresponds to a topic. The relationships between the avatars and the cells define the preference matrix: an avatar (overlay node) is interested in all cells (topics) intersecting with its vision circle. **Preference type II** is generated in a similar way except that the avatars linger at several dense-population areas in the virtual world instead of being randomly distributed.⁵ This causes the avatars in the same dense-population area to share lot of common preferences. An example for the generation of preference type I and type II is shown in Figure 21.

Since the end-to-end overlay latency is usually not less than the IP layer network distance between the same pair of nodes, we use the difference between the overlay latency and the network distance to measure the latency quality of the overlay topologies. Here, we use two metrics: one is *latency penalty*, which equals the overlay latency minus the network distance; the other is *latency stretch*, which is the ratio of the overlay latency to the network distance (the same definition is also used in [89]). Both metrics reveal the same trend but they offer two different representations.

⁵Note that the distance between avatars in the virtual world has nothing to do with the network distance between their corresponding overlay nodes.

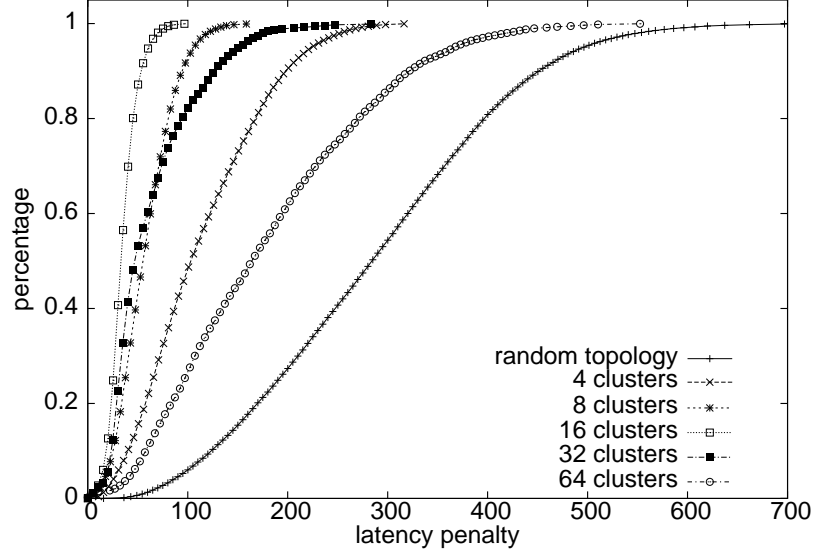


Figure 22: CDF of Latency Penalty When $\omega = 0$

5.4.2 Impact of Clustering on Latency and Waste

In the experiments, we observe how clustering affects the latency and waste in the resulting overlay topologies. We start from the extreme cases where the value of clustering parameter ω (in Eq. 10) equals 0 or 1, and then we show the experiment results when ω varies between 0 and 1.

First, we observe how the latency is affected by the clustering. Figure 22 shows how the latency penalty varies with the number of clusters when $\omega = 0$. It shows that all the topologies constructed with clustering (into 4 to 64 clusters) have smaller latency penalty than the random topology, and clustering into 16 clusters has the smallest latency penalty. This shows that more clusters do not necessarily result in smaller latency penalty.

Figure 23 and Figure 24 show how the waste is affected by the clustering. I and II plot the waste penalty for preference type I and II respectively when $\omega = 1$. In II, clustering dramatically reduced the waste penalty compared with that of random topology; while in I, the waste penalty is only slightly smaller than that of random topology. This reveals that clustering is most beneficial for waste reduction under heterogeneous preference (like preference type II). In later discussion, all figures are based on the preference type II.

Tuning the value of ω changes both the latency and waste in the resulting overlay

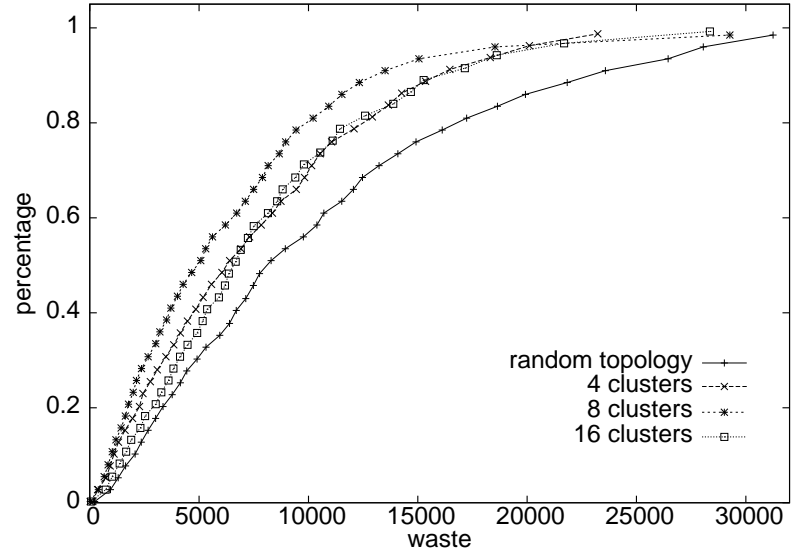


Figure 23: CDF of Waste for Preferences Type I When $\omega = 1$

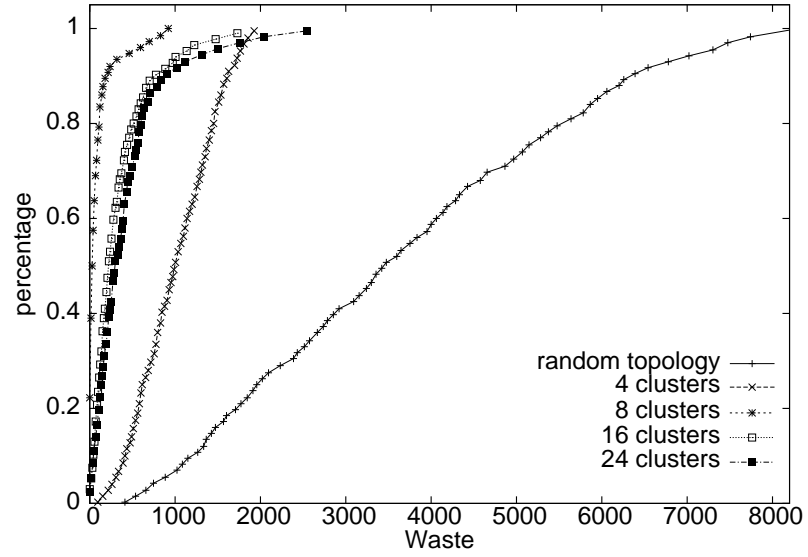


Figure 24: CDF of Waste for Preferences Type II When $\omega = 1$

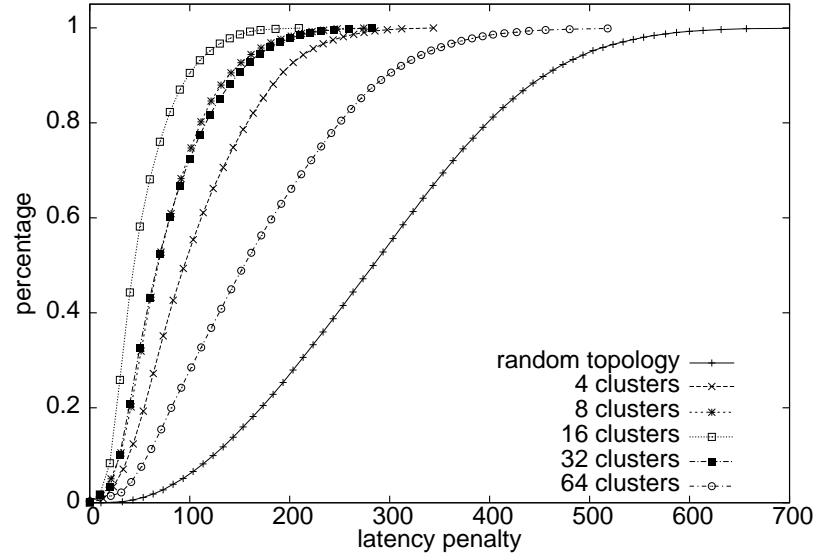


Figure 25: CDF of Latency When $\omega = 0.5$

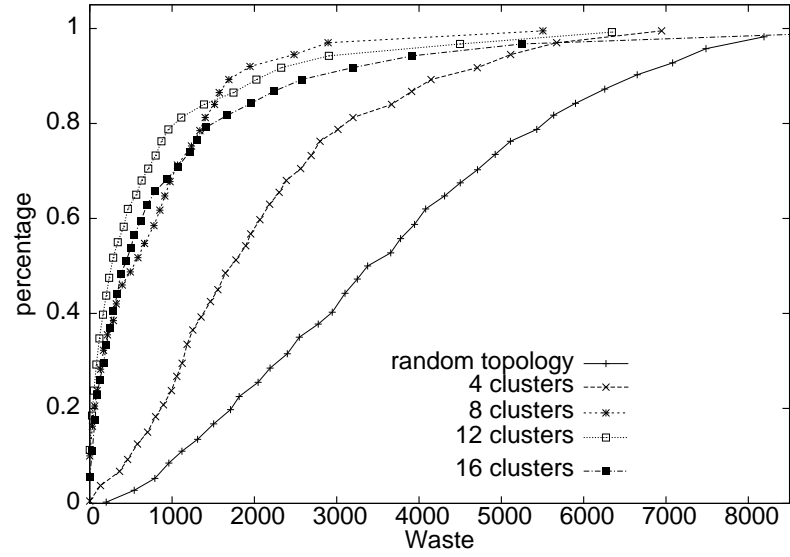


Figure 26: CDF of Waste When $\omega = 0.5$

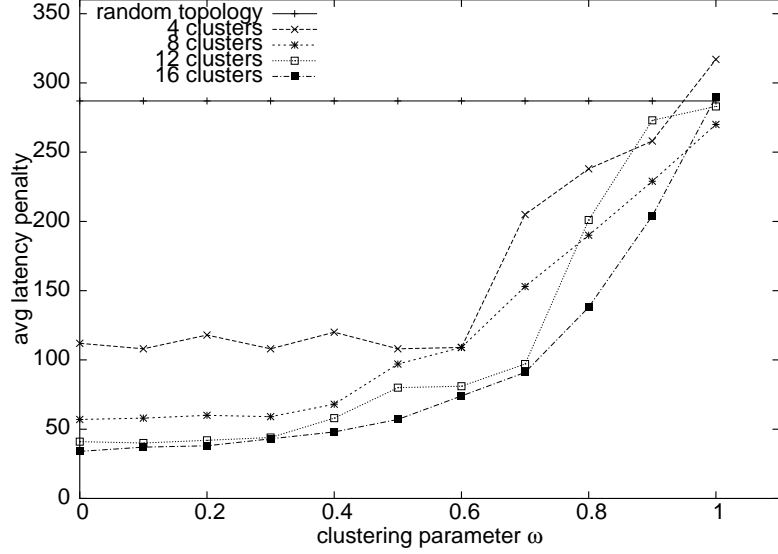


Figure 27: Latency Affected by Clustering Parameter ω

topology. Figure 25 and Figure 26 show the variation of latency and waste with the number of clusters respectively when $\omega = 0.5$. Comparing Figure 25 with Figure 22, the latency penalty is increased when ω is increased from 0 to 0.5; comparing with Figure 26, the waste penalty is increased when ω is decreased from 1 to 0. Generally, we observe that the larger the value of ω is, the less the waste and the longer latency the resulting overlay topologies have, and the vice versa. These trends are shown in Figures 27 and Figure 28 for latency penalty and waste penalty respectively.

By tuning the value of ω when constructing overlay topologies, the application can make a tradeoff between latency and waste in the resulting overlay topology. Figure 29 shows the combined value of overall latency and overall waste when the application specifies $\lambda = 0.5$ (as defined in Eq. 8). The overall latency and the overall waste are normalized to be within range $[0, 1]$ before being combined together. Under this requirement, it shows that the combined objective function is minimized when $\omega = 0.7$ and the number of clusters is 16. Practically, when constructing the optimal overlay topology for a given value of λ , the optimal ω and the number of clusters can be estimated through sampling. The accuracy of the estimation depends on the number of samples. In our experiments, we find that the objective function follows a clear trend (often convex, e.g., in Figure 29) and allows for

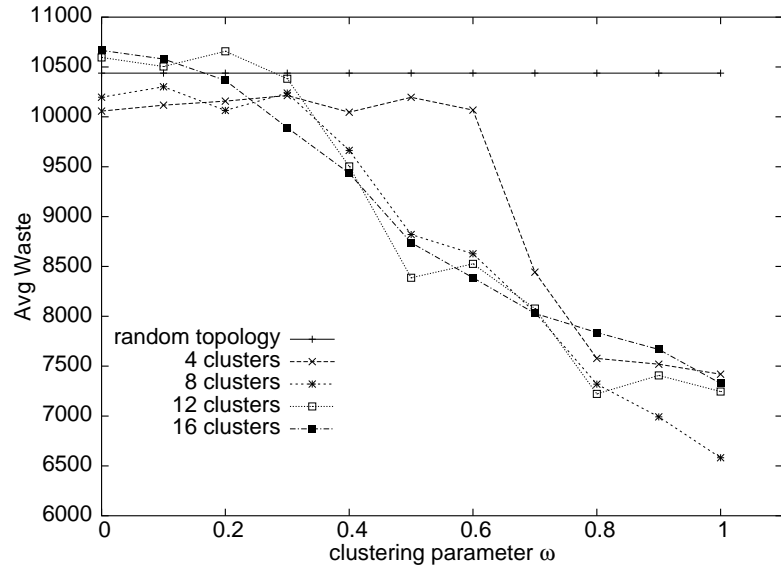


Figure 28: Waste Affected by Clustering Parameter ω

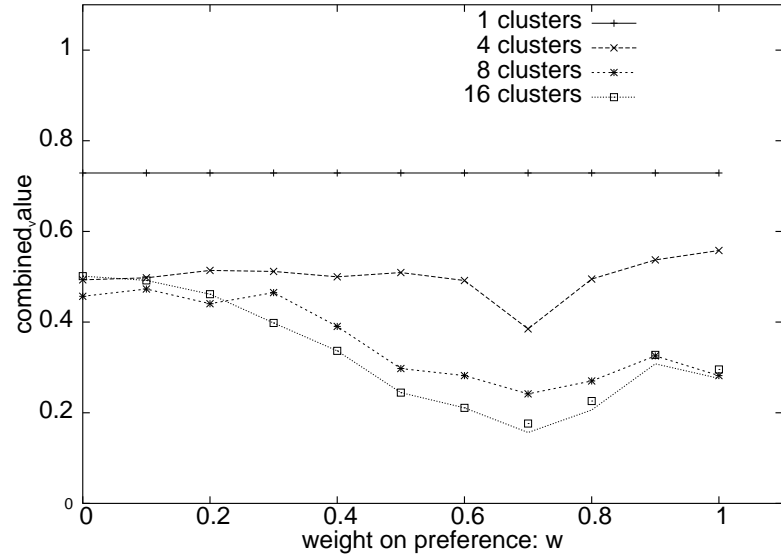


Figure 29: Combination of Latency and Waste When $\lambda = 0.5$

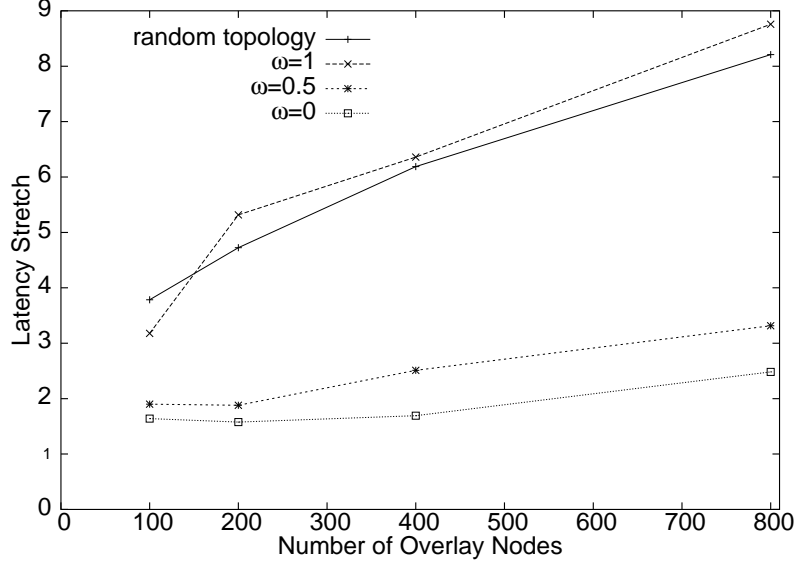


Figure 30: Latency Versus Size of Network

more efficient methods of estimation, e.g., sampling with a large granularity first and then fine-tuning in smaller scope.

5.4.3 Impact of Overlay Network Scale on Latency and Waste

In this section, we want to reveal how our algorithm performs when the scale of the overlay network becomes large. We increase the scale of the overlay network by including more underlying IP layer nodes into the set of overlay nodes and compare the performance of the random topology with three topologies that are built using our algorithm with different ω and their corresponding optimal number of clusters.

Figure 30 shows the latency stretch variation versus the size of overlay network. When the clustering parameter ω equals 1, clustering does not consider network proximity at all so the resulting topology does not have any advantage in term of latency over the random topology. As the clustering parameter ω decreases, the average latency stretch decreases and becomes much lower than that of the random topology.

Figure 31 shows that the variation of the average waste penalty versus the size of overlay network for the same four overlay topologies. When $\omega = 0$, which means the algorithm considers only the network proximity, the average waste is similar to that of the random

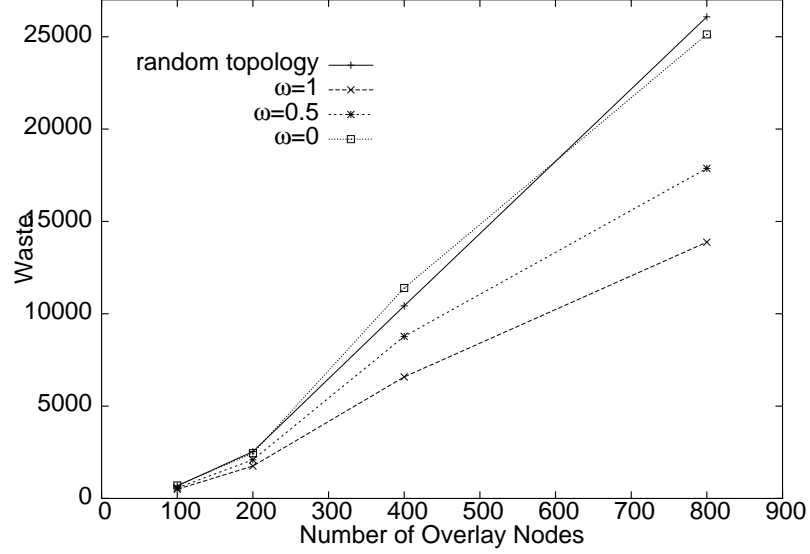


Figure 31: Waste Versus Size of Network

topology. As ω increases towards 1, the average waste penalty is considerably reduced compared with random topology.

The experiments show that our algorithm performs well in large scale overlay network and can help to achieve a tradeoff between the latency and the waste.

5.5 Summary

In this chapter, we have studied the problem of constructing preference-aware overlay topology for applications that are based on group communication model and have heterogeneous users' preferences. We have shown that overlay topologies constructed with only latency in mind may force users to receive and relay data they do not prefer and cause waste on users' networking and computation resources. When choosing the topology for an overlay network, therefore, a tradeoff between the overall latency and the total amount of waste data has to be made. For this end, we presented a heuristic algorithm for building overlay topologies with both the users' preferences to data and the network proximity in mind. In our algorithm, instead of building the overlay topology and dealing with all the complexity in one single step, we decomposed the process into two steps: clustering the nodes first based on both their network distances and the similarities between their preferences, and

then adding intra-cluster and inter-cluster overlay links based on the clustering. Our simulation results showed that our algorithm has good performance. They also showed that our algorithm is tunable to satisfy the need of different applications that desire different level of tradeoff between the latency and the amount of waste data.

CHAPTER VI

GT-P2PRMI

6.1 *Introduction*

Middleware systems that support the client-server paradigm have considerably reduced the complexity of building distributed applications. The remote method invocation (RMI) facility provided by such systems allows a client application to access remote services using the familiar procedure call abstraction. Although the client-server paradigm reduces programming complexity, several other issues such as locality, scalability and fault-tolerance remain to be addressed satisfactorily in such systems.

Peer-to-peer (P2P) systems have recently become highly popular. Collectively, such systems have large amount of resources which grow as demand for the resources grows because the new nodes that create additional demand also contribute new resources to the distributed system. Several projects have tried to utilize resources in such systems to provide services such as file systems(e.g. Farsite [11]). In this chapter, we explore how generic services, where clients use an RMI facility to access the services, can be implemented to exploit the resources that are offered by P2P systems. Since we want to retain the benefits of the RMI facility, we must enhance distributed object middleware to support service access at peer nodes where the service may be available.

Java RMI is widely used in distributed applications. Its platform independence makes it attractive for cooperation among heterogeneous peer nodes. It is desirable to extend an abstraction like Java RMI to peer-to-peer systems so applications can be built in a way similar to other distributed system.

P2P implementations of RMI present several new challenges. First, unlike dedicated server nodes, peer nodes may not always be present and can even refuse to provide service. Second, new service instances must be created dynamically either when new peers join the system or when existing peers with service instances leave the system. Such dynamic change

in the peers where the service may be available requires that a peer that is interested in the service must be able to locate it. Also, when no instances can be found close to the peer, it is necessary to dynamically create a new instance of the service at the requesting peer. The problem of finding nodes that provide a service has received considerable attention in P2P systems [6, 98].

In this chapter, we use the Java RMI framework to explore P2P implementations of distributed object-based middleware. In particular, we enhance the middleware to locate and invoke a service that is implemented by a dynamically varying set of peer nodes. If a peer refuses to provide service or a peer that previously provided the service is no longer in the system, our framework transparently redirects the remote invocation to another peer or creates a new service instance based on a policy that can be specified by the system. We evaluate the effectiveness of this approach using an implementation as well as a simulation of a large scale P2P system. Our preliminary results show that it is possible to harness the resources in P2P systems to deal with scalability and fault-tolerance problems even when a small fraction of peer nodes run instances of a service.

6.2 GT-P2PRMI: An Extension of Java RMI Framework

In this section, we describe the interfaces and implementation of GT-P2PRMI, which is an extension of the Java RMI framework. Compared to the traditional Java RMI, GT-P2PRMI implements several new functions to support a dynamic peer-to-peer environment:

- **On-Demand Service Replication:** At the time a node invokes an object, GT-P2PRMI allows the node to instantiate a replicated service instance locally to provide closer service not only to itself but also to clients in its vicinity. If good replication policies are adopted to control the instantiation of service replicas, GT-P2PRMI can help achieve application-level performance objectives such as better load balancing and shorter response time for RMI invocations.
- **Transparent Server Selection:** When a client makes an RMI invocation, GT-P2PRMI selects the server that might provide the better response latency (e.g., the

one with lightest load and shortest round-trip time), and transparently directs the RMI invocation to the service instance at such a server node.

- **Transparent Failure Handling:** In GT-P2PRMI, when an RMI invocation returns exceptions (e.g., the peer running the service is overloaded and refuses to accept new requests, or the service has been terminated at this), GT-P2PRMI will retry on other peers until it finds one that can process the RMI request. From the clients' perspective, the probability of service failure is much lower since it sees an exception only if all the peers that provide the service fail or refuse to accept the request.
- **Seamless Backward Compatibility:** GT-P2PRMI is an extension of traditional Java RMI and it provides all its functionality. Server and client programs that are aware of GT-P2PRMI can fully exploit its power while those that choose not to be aware of it can treat it like traditional RMI environment.

6.2.1 User Interface of GT-P2PRMI

For backward compatibility, GT-P2PRMI extends but does not change the traditional Java RMI interface so that client and server programs written for Java RMI can also run within the GT-P2PRMI framework without any change.

In the peer-to-peer environment, the client machines are actually servants: they request services from other peers that provide services, and can also potentially replicate some services and provide them to others. To take advantage of the new framework, instead of running *rmiregistry* of Java RMI, each servant node runs *p2prmiregistry*, which is an extension of the traditional *rmiregistry* with the additional ability to publish and look up replicated service instances in the peer-to-peer system. The client and server programs do not need be changed to take advantage of the new framework except that they now consult the local *p2prmiregistry*. Last, the replication policies are stored in a policy file. The client program retrieves the file name together with other system properties from the command line when it starts.

6.2.2 Implementation of GT-P2PRMI

As mentioned earlier, in the design of GT-P2PRMI, we tried to maintain the same server and client programming interface. Therefore, the major new functions of GT-P2PRMI are implemented by substantial enhancement of the *rmiregistry* daemon and an addition of a *delegate* at the client side.

6.2.2.1 P2PRMIRegistry: An Extension of RMIRegistry

First, the *p2prmieregistry* is used to form an overlay network for all peer nodes that start *p2prmieregistry* for a certain kind of service. This is achieved by having the *p2prmieregistry* running on each peer node to setup and maintain connections with each other. The neighbor to connect with depends on the underlying overlay network structure.

Second, the *p2prmieregistry* is also responsible for publishing and finding available services on the peer network. The peers collectively maintain a global directory of all services provided by all nodes in the peer network. The *p2prmieregistry* uses two functions, *publish()* and *search()*, as the common interface to operate on the global service directory; the former function advertises a service to the peer network and the latter locates a list of nodes that provide this service. The implementation of *publish()* and *search()* depends on the structure of the underlying overlay network. GT-P2PRMI provides three different implementations for *search()* and *publish()* that allow various types of searching and publishing methods to be used. The first one is for a central directory based peer network (e.g, as in Napster) where the global service directory is maintained at a well-known central location. The second one is for structured peer network, where the global service directory is maintained by all the peers using Chord's distributed hash table (DHT) scheme [98]. The third one is for an unstructured peer network, organized like Gnutella [6] does. In this implementation, each node maintains a directory of its own services while the searching messages are forwarded to neighbors with controlled flooding.

The *p2prmieregistry* provides functions *p2pbind()* and *p2plookup()* to facilitate the implementation of the *Naming* class used by server and client programs.¹ The following is a

¹For backward compatibility, the *p2prmieregistry* also implements the *bind()* and *lookup()* functions of

brief description of these functions.

1. **p2pbind**(service-name, service-object): When a server program calls *Naming.bind()*, the *Naming* class invokes the *p2pbind()* function to register the service. After the normal processing associated with standard *bind()*, the *p2pbind()* function also publishes the newly available service instance to the peer network by calling *publish()*. (The same thing is done for *rebind()*).
2. **p2plookup**(service-name): When a client program calls *Naming.lookup()*, the *Naming* class invokes the *p2plookup()* function on the *p2prmiregistry* to locate the service. The *p2plookup()* function will search the peer network using *search()* and return a list of peer nodes that provide the requested service.

6.2.2.2 Service Invocation Wrapping

The *lookup()* function in the *Naming* class is implemented as follows. When the client calls *Naming.lookup()* to locate a service, it invokes the *p2plookup()* function on the *p2prmiregistry* to retrieve a list of nodes providing the service. It then selects one node from the list and invokes the *lookup()* function on the *p2prmiregistry* running on the selected node to retrieve the stub object of the service. The *Naming* class constructs the *p2pdelegate* object based on the stub object and returns the *p2pdelegate* object to the calling client. Currently, when selecting a node from the list, we use ping message to test response time of peers that provide the service and choose the one with minimum response time. More complex server selection schemes [45, 91] can be adopted into GT-P2PRMI.

The *p2pdelegate* object is designed as a wrapper of the normal stub object and has the same interface as the stub object. When invoked by the client, it will invoke the service on the selected server using the normal stub object and retry other servers when there is a failure. Details of service invocation, including failure handling, are shown in Figure 32.

This figure shows that peer node *B* has started the service, node *A* is just starting the service and node *C* is requesting service first from node *B*. The invocation execution

rmiregistry to handle requests from server/client programs that were compiled with standard Java RMI library.

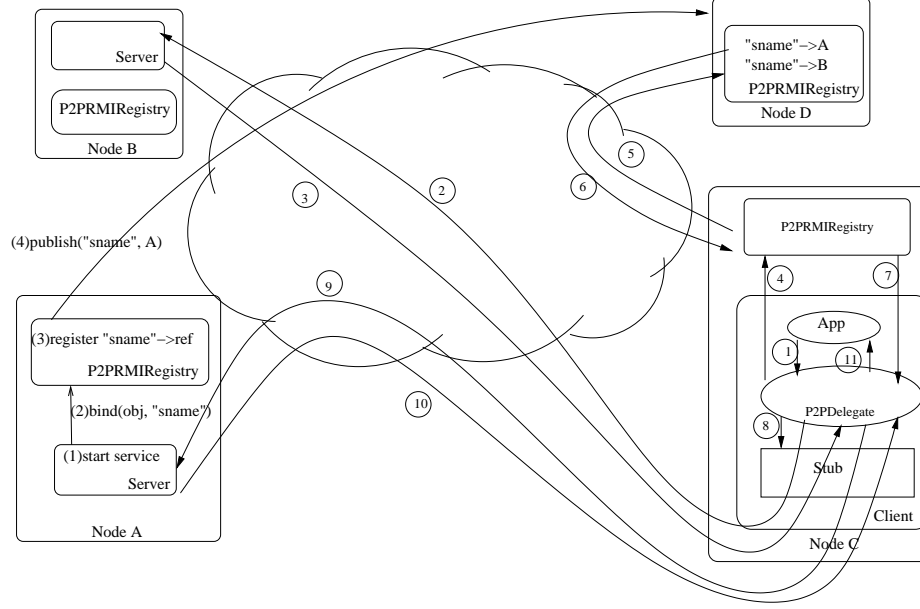


Figure 32: Illustration of Remote Invocation in GT-P2PRMI

steps for node *C* are as follows: 1. client sends `invoke()` request; 2. `invoke()` is sent to peer node *B*; 3. invocation failed at *B*; 4. *C* sends request to its *p2prmireregistry* to search for available peers that provide service with name "sname"; 5. *p2prmireregistry* searches through the P2P overlay network; 6. *C* receives list of service providing peers; 7. list of service providing peers is returned to *p2pdelegate*; 8. *p2pdelegate* selects peer *A* and downloads the stub from node it to replace the old one; 9. invocation request is sent to node *A*; 10. *C* receives result from node *A*. The *p2pdelegate* object is constructed with additional code for service replication. At each invocation, the *p2pdelegate* object makes a decision about whether to replicate the service locally and share it with other peers. This decision is made at runtime by consulting the policy file. If it decides to replicate the service, then it downloads the related objects (including skeleton and stub) from a peer that currently provides the service and then instantiates the service locally. Object caching and consistency maintenance schemes described in [62, 61] are used to ensure that consistency among the various replicated instances is maintained.

By allowing more peer nodes to contribute their resources by providing a service via dynamic replication, service response time can be reduced if service instances are created at

appropriate locations. Our framework supports different replication policies and the peers can define their own policies. The replication decision is made at each peer locally and is not coordinated by a central node. For example, the replication policy encouraging fair contribution of resources by all peers could be: a peer node replicates a service with the probability of $r^{1/d}$, where r is a value between 0 and 1 set by peer, d is a normalized value of the distance from the peer node to the nearest node that runs an instance of the same service. This policy limits too many instances in the same neighborhood of peers and the number of instances can be controlled by setting r to an appropriate value. Other more complex replication algorithms, e.g., AURA, can also be used through more complicated implementation.

6.3 Performance Studies

We have built a prototype of the *P2PRMI* framework. We used this prototype to measure the time spent in searching, replicating, and local and remote invocation within this framework. We simulated a larger scale system that models service replication and sharing among peer nodes with our distributed replication policy. The simulation shows how invocation response time varies as we change replication parameters.

6.3.1 Experimental Evaluation in Small Scale Environment

We measured the invocation time of standard Java RMI within a LAN environment. The invocation time increases with number of clients that concurrently send requests. This increase is linear initially since the clients' requests are waiting in a queue to be served. As the number of clients increases beyond a number, the degradation in response time will be more significant.

We also used 10 machines from University of Utah and Georgia Tech to run our GT-*P2PRMI* with Chord DHT implemented for locating service instances. Only one type of service was run in the system. All the machines that start the service send their advertisements to machine *A* whose hash value of its IP address is closest to the hash value of the service name. The average time spent on finding the a service instance from machine *A* is 42ms when all the nodes on the searching path are in the same LAN of GaTech and 210ms

when there is one hop from Utah to GaTech. The average number of hops is 3.5. After finding the node with the service instance, the invocation time is 2.07ms if the found service provider is in the same LAN, and 49.1ms if the found service provider is at Utah (client is at GaTech). Also, the time spent on replicating the service from a node at the same LAN is about 20ms, and is about 110ms when from a remote node that is at Utah (client is at GaTech).

6.3.2 Simulation of Large Scale Internet Environment

We use GT-ITM [114] to generate a topology of a larger scale network with 1640 IP level nodes. We randomly choose 110 end nodes from the stub domains. We also use the end-to-end latency between any two nodes of the 110 nodes under the generated topology with GT-ITM. To simplify the simulation, we randomly choose one node as a central naming server for finding service instances. All requests for locating service instances are sent to the naming server. All advertisements for replicating a service are also sent to the naming servers. One node is randomly chosen as the first node that is providing the service.

All nodes start to generate service requests with exponential distribution with parameters λ . This means the interval of previous request returning and generation of next request follows the exponential distribution. After locating potential service providers, the peer node that generates the request can also decide if it wants to replicate and share this service with a probability of $r^{1/d}$. d is the distance to the nearest peer that runs an instance of the service. The invocation time is the sum of lookup time (if the invoking peer wants to search for better service provider), round-trip time to peer that executes the invocation and the queuing time at the server. In Figure 34, x and y axis represent the same parameters as Figure 33. We can see that the rate of searching for a better service instance can also impact the invocation time. When the rate is one search every 20 invocation, the average latency is the best; when the rate is one search every 200 invocations or one search every 5 invocations, both cases experience a larger latency for completing an invocation. In Figure 33, the x axis is the probability that specifies when each node will choose to replicate and share a service instance for a period of time (it is based on r and d). The y axis is the

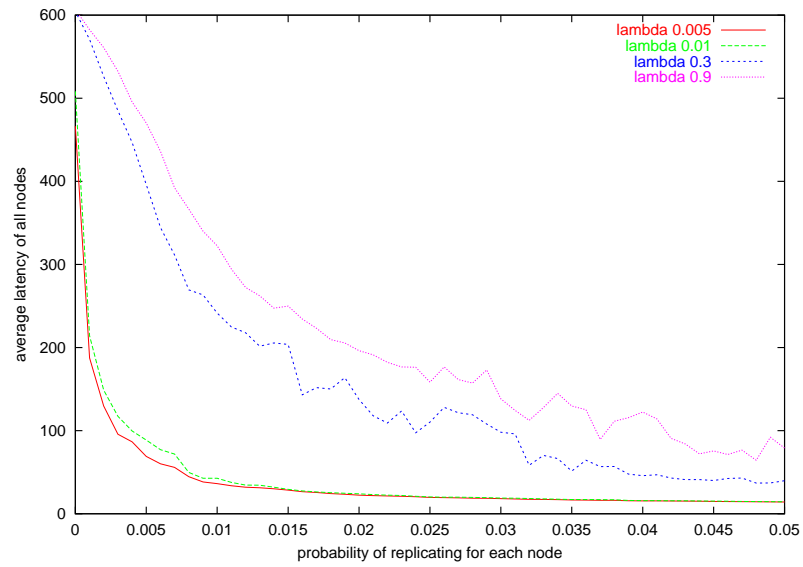


Figure 33: Invocation Latency Affected by Replication Probability

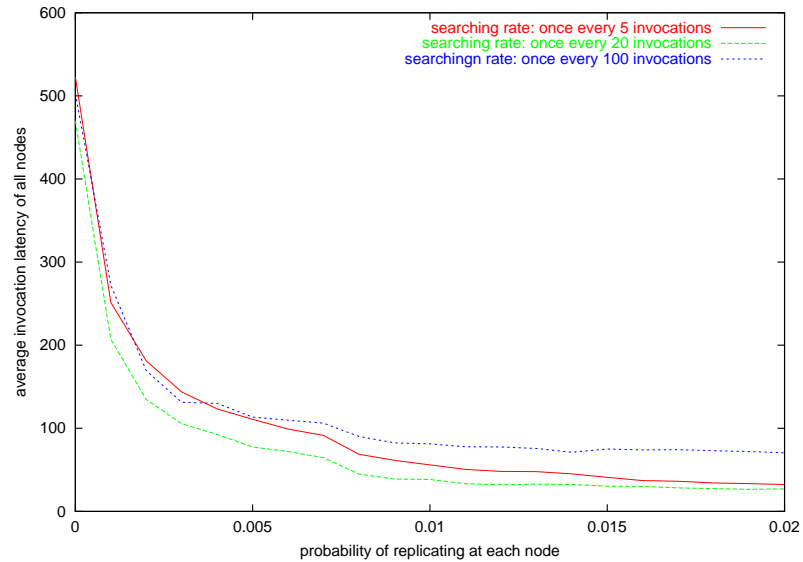


Figure 34: Invocation Latency Affected by Lookup Frequency

average invocation latency over all of the 110 peer nodes. As the probability of replicating a service is increased to 0.01, the average invocation latency is decreased dramatically. In our simulation, each peer runs the service for $200 * 1/\lambda$ ms before shutdown since we assume that service provided by a peer node can be terminated at any time. Thus, when the replication probability is 0.01, the average number of service instances increases to 10 among the 110 nodes, and it increases to 16 when replication probability is 0.02. Figure 34 also shows that as the λ increase, the invocation latency is also increased since more service request are generated at each node.

6.4 *Summary*

We have explored implementation of distributed object-based middleware in peer-to-peer environments. In such systems, peer nodes can create new replicated instances of a service as the size of the system grows. A peer can either access a service instance in its vicinity or can create a local instance when a instance that is close does not exist. We extended the Java RMI framework to explore these ideas by building a prototype. Our experimental and simulation results show the need for such a design and the benefits that are offered by it.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

As computing devices become pervasive and better connected, the scalability requirements for Internet-based services are also increasing. Distributed object middleware has been widely used to develop such applications since it made it easier to rapidly develop distributed applications for heterogeneous computing and communication systems. However, the quality of service experienced by users could deteriorate when the demands for the service increase while the service capacity does not adapt accordingly. We developed the distributed object middleware that can dynamically find and utilize distributed resources available at all the participating users; therefore, the service response time experienced by each user is not affected by the heterogeneous distribution of users and their demands.

7.1 *Contributions*

We explored a number of research issues to validate our thesis that non-dedicated resources in a distributed system can be utilized to replicate shared objects dynamically so that the quality and scalability of a distributed service can be achieved with lower cost by replicating the objects at right places and by disseminating updates to those shared objects efficiently and quickly. The following are the contributions of our work that has been done to validate the thesis.

- **P2P Replication** We have developed a new *fair and self-managing replication algorithm* that allows distributed non-dedicated resources to be used to improve service performance with lower cost. It has the following desirable characteristics:
 - Stable Replica Fraction: The algorithm can maintain the number replicas in a stable replica fraction range.
 - Autonomous Decision Making: There is no central server to make replication

decision for any peer nodes. Each peer node makes the decision about whether it should create a replica based only on its locally maintained information.

- Fair Resource Contribution: Each peer node replicates a service for a similar amount of time.
- Low Response Time: The service response time is low because replicas are created at right places.

- **Efficient Overlay Network Generation and Multicast Communication** We

developed multicast grouping algorithm and overlay construction algorithms for disseminating updates among a set of replicating nodes. Both algorithms aim at reducing the network bandwidth usage and minimize the latency of disseminating data among the set of interested receivers, especially users with heterogeneous interests. A **multicast grouping algorithm** was developed to disseminate updates to the shared objects among a large set of heterogeneous peer nodes to keep consistent view for all peer nodes. The algorithm groups nodes with similar interests into same group and multicasts all the required data to the group so that the unwanted data received by each node can be minimized. An **overlay construction algorithm** reduces both network latency and total network traffic when delivering data through the built overlay network. The multi-attribute clustering algorithm can be tuned to adapt to application user's desired tradeoff between network latency and traffic (which implies the amount of money charged).

- **GT-P2PRMI** We implemented a distributed object framework, GT-RMI, that allows a host machine to instantiate a replicated service instance locally to provide closer service not only to itself but also to clients in its vicinity. It also allows peer nodes to invoke dynamically replicated objects transparently. The user only needs to specify the desired service, the requests will be directed to selected replicas. If the invocation fails, it will be redirected to other available service provider. Any peer node can replicate the object and advertise it to the public. The framework can be configured for a particular peer node through a policy file. GT-P2PRMI is an extension of

traditional Java RMI and it provides all its functionality. Server and client programs that are aware of GT-P2PRMI can fully exploit its power while those that choose not to be aware of it can treat it like traditional RMI environment.

7.2 *Future Work*

Several research issues still need to be explored to realize a fully distributed middleware framework that can adapt to a heterogeneous request pattern and resource availability peer-to-peer environment. We will briefly discuss some of the more interesting problems.

- **Economical Model of Resource Contribution** Currently, we assume that peer nodes are not selfish and they faithfully execute the specified protocols to contribute their resources to the system. However, in a peer-to-peer environment where there is no central control and a selfish peer node may want to maximize its own benefit. The actions of each peer node should be motivated by incentives for contributing resources for others. The amount of resources each peer node contributes should be decided by the amount of services it can get. Thus, one possible extension to our replication scheme is we can introduce the notion of *currency* which can be used to buy and sell the resources. How best to manage such currency for replication of generic service presents a number of challenges that remain to be addressed.
- **Distributed Decision Making in Updates Dissemination** Currently, we need to collect the information, i.e., each node's preference, into a central location and make the grouping decision with the complete knowledge. Although a decision with global knowledge is more accurate, it is also more costly. One alternative is to have peer nodes exchange information with nearby peer nodes and make decision based on the partial information collected from local area. This approach may be able to incur much lower overhead with some sacrifice of performance. Such decentralized algorithms need to be explored in the future.
- **More Comprehensive Simulation Model** Our simulations do not consider network layer detail, such as congestion, failure on the transmission path, variation of

routing time between a pair of nodes. They do not take into account the effect of background competing traffic either. Therefore, we can not model the actual latency and bandwidth of the network. Future work should address these limitations by constructing a more comprehensive simulator with the capability of simulating the actual network traffic. Similarly, additional experimental results are needed for validating such simulation results.

- **Extension to GT-P2PRMI** In the future, we would like to explore several issues related to our GT-P2PRMI. We would explore how peer-to-peer protocols can be developed to maintain consistency of replicated instances of a service when the service state is updated by invocations. We will also explore security and trust issues to deal with potentially uncooperative and malicious peer nodes. Finally, we will do detailed evaluation of our approach to determine its effectiveness.

REFERENCES

- [1] “Jms:java message service(jms), <http://java.sun.com/jms/>,” in *Sun Microsystems*.
- [2] “Kazaa,” in <http://www.kazaa.com/us/index.htm>.
- [3] “Napsters,” in <http://www.napster.com>.
- [4] “Seti@home,” in <http://setiathome.ssl.berkeley.edu/>.
- [5] “Xml-rpc,” in <http://www.xmlrpc.com/>.
- [6] “Gnutella,” in <http://gnutella.wego.com/>, *Open Source Community*, 2001.
- [7] “The gnutella protocol specification v0.4.” <http://www9.limewire.com/developer/gnutella-protocol-0.4.pdf>, 2002.
- [8] A. CARZANIGA, D. S. R. and WOLF, A. L., “Achieving scalability and expressiveness in an internet-scale event notification service,” in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC 2000)*, Portland, Oregon, pp. 219-227, July 2000.
- [9] ADYA, A., BOLOSKY, W. J., CASTRO, M., CERMAK, G., CHAIKEN, R., DOUCEUR, J. R., HOWELL, J., LORCH, J. R., THEIMER, M., and WATTENHOFER, R. P., “Farsite: federated, available, and reliable storage for an incompletely trusted environment,” in *SIGOPS Oper. Syst. Rev.*, vol. 36, (New York, NY, USA), pp. 1–14, ACM Press, 2002.
- [10] ADYA, A., BOLOSKY, W. J., CASTRO, M., CERMAK, G., CHAIKEN, R., DOUCEUR, J. R., HOWELL, J., LORCH, J. R., THEIMER, M., and WATTENHOFER, R. P., “Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs,” in *OSDI*, 2002.
- [11] ADYA, A., BOLOSKY, W. J., and ET AL., “Farsite: Federated, available, and reliable storage for an incompletely trusted environment,” *5th Symposium on Operating Systems Design and Implementation*, 2002.
- [12] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., and MORRIS, R., “Resilient overlay networks,” in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-01)*, (New York), pp. 131–145, 2001.
- [13] ARMITAGE, G., “An experimental estimation of latency sensitivity in multiplayer quake 3,” in *In Proceedings of the 11th IEEE International Conference on Networks (ICON 2003)*, Sydney, Australia.
- [14] AUENCA-ACUNA, F. M., MARTIN, R. P., and NYUYEN, T. D., “Autonomous replication for high availability in unstructured p2p systems,” in *22nd International Symposium on Reliable Distributed Systems (SRDS)*, 2003.

- [15] BAENTSCH, M., BAUM, L., MOLTER, G., ROTHKUGEL, S., and STURM, P., “Enhancing the webs infrastructure: From caching to replication,” in *IEEE Internet Comput.*, 1(2):18C27, Mar. 1997.
- [16] BALLARDIE, T., “Core based tree (cbt) multicast – architectural overview and specification,” 1995.
- [17] BALLINTIJN, G., VAN STEEN, M., and TANENBAUM, A., “Exploiting location awareness for scalable location-independent object ids,” in *Proc. Fifth Annual ASCI Conf, Heijden, The Netherlands*, pp. 321–328, June 1999.
- [18] BANERJEE, S., BHATTACHARJEE, B., KOMMAREDDY, C., and VARGHESE, G., “Scalable application layer multicast,” in *Proceedings of the ACM SIGCOMM 2002*, (New York), pp. 205–220, 2002.
- [19] BARATLOO, A., CHUNG, P. E., HUNANG, Y., RANGARAJAN, S., and SHALINI YAJNIK, “Filterfresh: Hot replication of java rmi server objects,” in *Object-Oriented Technologies and Systems (COOTS)*, April 1998.
- [20] BECK, M., MOORE, T., and PLANK, J. S., “An end-to-end approach to globally scalable programmable networking,” in *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, ACM Press, 2003.
- [21] BHATTACHARJEE, B., KWON, S. B., and FAHMY, S., “Scalable application-layer multicast for content distribution,” *SIGCOMM*, August 2002.
- [22] BROWN, N. and KINDEL, C., *Distributed Component Object Model Protocol – DCOM/1.0*. Microsoft Corporation, Redmond, WA, 1996.
- [23] BUCHHOLZ, S. and BUCHHOLZ, T., “Replica placement in adaptive content distribution networks,” in *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1705–1710, ACM Press, 2004.
- [24] CALVIN, J. O., CHIAN, C. J., and HOOK, D. J. V., “Data subscription,” in *DIS Workshop*, 1995.
- [25] CARTON, B. and MESAROS, V., “Improving the scalability of logarithmic-degree dht-based peer-to-peer networks,” in *Euro-Par*, 2004.
- [26] CARZANIGA, A., R. D. S. and WOLF, A. L., “Design and evaluation of a wide-area event notification service,” in *CM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332C383, 2001.
- [27] CHANKHUNTHOD, A., DANZIG, P., NEERDAELS, C., SCHWARTZ, M., and WORRELL, K., “A hierarchical internet object cache,” tech. rep.
- [28] CHAWATHE, Y., “Scattercast: an adaptable broadcast distribution framework,” in *Multimedia System*, vol. 9, no. 1, pp. 104C118, 2003.
- [29] CHEN, Y., KATZ, R. H., and KUBIATOWICZ, J. D., “Dynamic replica placement for scalable content delivery,” 2002.
- [30] CHIUH, T., “Resource virtualization techniques for wide-area overlay networks,” tech. rep., State University of New York at Stony Brook.

- [31] CHU, Y.-H., RAO, S. G., and ZHANG, H., "A case for end system multicast," in *ACM SIGMETRICS 2000*, (Santa Clara, CA), pp. 1–12, ACM, June 2000.
- [32] CLARKE, I., SANDBERG, O., WILEY, B., and HONG, T. W., "Freenet: A distributed anonymous information storage and retrieval system," *Lecture Notes in Computer Science*, vol. 2009, pp. 46+, 2001.
- [33] COHEN, E. and SHENKER, S., "Replication strategies in unstructured peer-to-peer networks," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 177–190, ACM Press, 2002.
- [34] D. INGHAM, M. LITTLE, S. C. and SHRIVASTAVA, S., "W3objects: Bringing object-oriented technology to the web," in *The Web Journal*, (1):89C105, 1995.
- [35] D. PENDARAKIS, S. SHI, D. V. and WALDVOGEL, M., "Almi: An application level multicast infrastructure," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS 01)*, (San Francisco, CA, USA), pp. 49C60, 2001.
- [36] D. TERRY, A. DEMERS, K. P. M. S. M. T. and WELSH, B., "Session guarantees for weakly consistent replicated data," in *In Proc. Third Intl Conf. on Parallel and Distributed Information Systems*, pp. 140C149, Austin, TX, Sept. 1994. IEEE.
- [37] DEERING, S., ESTRIN, D. L., FARINACCI, D., JACOBSON, V., LIU, C., and WEI, L., "Pim architecture for wide-area multicast routing," in *SIGCOMM*, 1994.
- [38] DONNELLEY, J., "Www media distribution via hopwise reliable multicast," in *Comp. Netw. ISDN Syst.*, 27(6):781C788, 1995.
- [39] DOUCEUR, J. R. and WATTENHOFER, R. P., "Large-scale simulation of replica placement algorithms for a serverless distributed file system," in *9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001.
- [40] EBERHARD, J. and TRIPATHI, A., "Efficient object caching for distributed java rmi applications," in *Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, (London, UK), pp. 15–35, Springer-Verlag, 2001.
- [41] ERIKSSON, H., "Mbone: the multicast backbone," *Commun. ACM*, vol. 37, no. 8, pp. 54–60, 1994.
- [42] ESTRIN, D., "Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture." draft-ietf-idmr-pim-arch-01.ps, Internet Draft.
- [43] EUGENE, T. S. and ZHANG, H., "Predicting internet network distance with coordinates-based approaches," in *Proceeding of INFOCOM*, 2002.
- [44] FEAMSTER, N., BALAKRISHNAN, H., REXFORD, J., SHAIKH, A., and VAN DER MERWE, J., "The case for separating routing from routers," in *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, ACM Press, 2004.

- [45] FEI, Z., BHATTACHARJEE, S., ZEGURA, E. W., and AMMAR, M. H., "A novel server selection technique for improving the response time of a replicated service," *IEEE Infocom*, 1998.
- [46] FINK, R., "Network integration — boning up on IPv6 — the 6bone global test bed will become the new Internet," *Byte Magazine*, vol. 23, Mar. 1998.
- [47] FRANCIS, P., "Yoid: Extending the internet multicast architecture," in <http://www.icir.org/yoid/docs/index.html>, April 2000.
- [48] GARREY, M. R. and HOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco: W.H.Freeman and Company, 1979.
- [49] GONG, L., "Jxta: A network programming environment," in *IEEE Internet Computing*, pp. 88–95, 2001.
- [50] GWERTZMAN, J. and SELTZER, M., "The case for geographical push-caching," in *In Proc. Fifth HOTOS, Orcas Island, WA, May 1996. IEEE*.
- [51] J. JANNOTTI, D. K. GIFFORD, K. L. J. and KAASHOEK, M. F., "Overcast: Reliable multicasting with an overlay network," in *Proceedings of OSDI 2000, pp. 197C212, 2000*.
- [52] JIN, Y. and STROM, R. E., "Relational subscription middleware for internet-scale publish-subscribe," in *Proceedings of the 2nd International Workshop on Distributed Event-based Systems(DEBS03), June 2003*.
- [53] KANGASHARJU, J., ROBERTS, J., and ROSS, K., "Object replication strategies in content distribution networks," in *Computer Communications*, 2002.
- [54] KARLSSON, M. and KARAMANOLIS, C., "Bounds on the replication cost for qos," tech. rep., Hewlett Packard Labs, July, 2003.
- [55] KARLSSON, M. and KARAMANOLIS, C., "Choosing replica placement heuristics for wide-area systems," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, (Washington, DC, USA), pp. 350–359, IEEE Computer Society, 2004.
- [56] KEROMYTIS, A. D., MISRA, V., and RUBENSTEIN, D., "SOS: Secure overlay services," in *Proceedings of the ACM SIGCOMM 2002*, (New York), pp. 61–72, 2002.
- [57] KO, B.-J. and RUBENSTEIN, D., "Distributed, self-stabilizing placement of replicated resources in emerging networks," in *11th IEEE International Conference on Network Protocols (ICNP)*, November 2003.
- [58] KORUPOLU, M., PLAXTON, G., and RAJARAMAN, R., "Placement algorithms for hierarchical cooperative caching," in *Journal of Algorithms*, 38(1), pp. 260–302, January 2001.
- [59] KRISHNAMURTHY, B., WILLS, C., and ZHANG, Y., "On the use and performance of content distribution networks," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 169–182, ACM Press, 2001.

- [60] KRISHNASWAMY, V., GANEV, I. B., DHARAP, J. M., and AHAMAD, M., "Distributed object implementations for interactive applications," in *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, (Secaucus, NJ, USA), pp. 45–70, Springer-Verlag New York, Inc., 2000.
- [61] KRISHNASWAMY, V., GANEV, I. B., DHARAP, J. M., and AHAMAD, M., "Shared state consistency for time-sensitive distributed applications," *Middleware*, 2000.
- [62] KRISHNASWAMY, V., WALTHER, D., BHOLA, S., BOMMAIAH, E., RILEY, G., TOPOL, B., and AHAMAD, M., "Efficient implementation of Java Remote Method Invocation (RMI)," *COOTS*, 1998.
- [63] KRISHNASWAMY, V., WALTHER, D., BHOLA, S., BOMMAIAH, E., RILEY, G. F., TOPOL, B., and AHAMAD, M., "Efficient implementation of java remote method invocation (rmi).," in *COOTS*, pp. 19–27, 1998.
- [64] KUBIATOWICZ, J., "Oceanstore: An architecture for global-scale persistent storage," in *In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 190–201, November 2002.
- [65] KURMANOWYTSCH, D.-I. R., *Omnix: An Open Peer-to-Peer Middleware Framework*. PhD thesis, Vienna University of Technology, Feb 2004.
- [66] KWON, M. and FAHMY, S., "Topology aware overlay for group communication," *International Workshop on Network and Operating System Support for Digital Audio and Video*, 2002.
- [67] LAKSHMINARAYANAN, K., STOICA, I., BALAKRISHNAN, H., and KATZ, R., "OverQoS: Offering Internet QoS Using Overlays," in *1st HotNets Workshop*, (Princeton, NJ), October 2002.
- [68] LETY, E., TURLETTI, T., and BACCELLI, F., "Cell-based multicast grouping in large-scale virtual environments," in *INRIA*, 1996.
- [69] LI, Z. and MOHAPATRA, P., "QRON: QoS-aware routing in overlay networks," 2003. IEEE JSAC, 2003, to appear.
- [70] LI, Z. and MOHAPATRA, P., "The impact of topology on overlay routing service," in *Proceedings of IEEE INFOCOM'04*, 2004.
- [71] LIEBEHERR, J. and BEAM, T. K., "Hypercaster: A protocol for maintaining multicast group members in a logical hypercube topology," in *NGC '99: Proceedings of the First International COST264 Workshop on Networked Group Communication*, pp. 72–89, Springer-Verlag, 1999.
- [72] LV, Q., CAO, P., COHEN, E., LI, K., and SHENKER, S., "Search and replication in unstructured peer-to-peer networks," in *16th ACM International Conference on Supercomputing (ISC)*, June 2002.
- [73] MACEDNIA, R., ZYDA, M. J., PRATT, D. R., BRUTZMAN, D. P., and BARHAM, P. T., "Exploiting reality with multicast group," in *Virtual Reality Annual International Symposium*, sep 1995.

- [74] MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BARHAN, P. T., and ZESITZ, S., *A Network Software Architecture for Large-Scale Virtual Environments*. doctoral dissertation, Naval Postgraduate school, monterrey, CA, june 1995.
- [75] MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BARHAN, P. T., and ZESITZ, S., “Npsnet: A multi-player 3d virtual environment over the internet.,” in *Symposium on Interactive 3D Graphics*, 1995.
- [76] MESAROS, V., CARTON, B., and ROY, P. V., “P2ps: Peer-to-peer development platform for mozart,” in *Second International Mozart/Oz Conference*, 2004.
- [77] MORSE, K. L., BIC, L., and DILLENCOURT, M., “Interest management in large-scale virtual environments,” pp. 52–68, february 2000.
- [78] OBJECT MANAGEMENT GROUP, *The Common Object Request Broker: Architecture and Specification*. Object Management Group, 2.5 ed., September 2001.
- [79] PADMANABHAN, V. N. and SRIPANIDKULCHAI, K., “The case for cooperative networking,” in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*.
- [80] PAVLIN RADOSLAVOV, R. G. and ESTRIN, D., “Topology-informed internet replica placement,”
- [81] PENDARAKIS, D., SHI, S., VERMA, D., and WALDVOGEL, M., “ALMI: An application level multicast infrastructure,” in *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, (San Francisco, CA, USA), pp. 49–60, Mar. 2001.
- [82] PETERSON, L., ANDERSON, T., CULLER, D., and ROSCOE, T., “A blueprint for introducing disruptive technology into the internet,” *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, 2003.
- [83] PETERSON, L., SHENKER, S., and TURNER, J., “Overcoming the Internet Impasse Through Virtualization,” in *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.
- [84] QIU, L., PADMANABHAN, V. N., and VOELKER, G. M., “On the placement of web server replicas,” in *INFOCOM*, pp. 1587–1596, 2001.
- [85] RABINOVICH, M. and AGGARWAL, A., “Radar: A scalable architecture for a global web hosting service,” in *In Proceedings of the 8th International World Wide Web Conference*, May 1999.
- [86] RABINOVICH, M. and AND, R. R., “A dynamic object replication and migration protocol for an internet hosting service,” in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1999.
- [87] RAK, S. J. and HOOK, D. J. V., “Evaluation of grid-based relevance filtering for multicast group assignment,” in *DIS workshop*, 1996.
- [88] RATNASAMY, S. and DRUSCHEL, P., “Storage mnagement and caching in past: A large-scale, persistent peer-topeer storage utility,” in *ACM SOSP*, October 2001.

- [89] RATNASAMY, S., HANDLEY, M., KARP, R., and SHENKER, S., “Topologically-aware overlay construction and server selection,” in *Proceedings of IEEE INFOCOM’02*, 6 2002.
- [90] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., and SHENKER, S., “A scalable content addressable network,” in *Proceedings of ACM SIGCOMM 2001*, 2001.
- [91] RATNASAMY, S., HANDLEY, M., KARP, R., and SHENKER, S., “Topologically-aware overlay construction and server selection,” *IEEE Infocom*, 2002.
- [92] REDUNDANCY, E. R., “Appears in proceedings of the 11th ieee international conference on network protocols (icnp 2003).”
- [93] RUBENSTEIN, D. and SAHU, S., “Can unstructured p2p protocols survive flash crowds?,” *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 501–512, 2005.
- [94] S. BHOLA, R. STROM, S. B. Y. Z. and AUERBACH, J., “Exactly-once delivery in a content-based publish-subscribe system,” in *Proceedings of the International Conference on Dependable Systems and Networks (DSN02)*, IEEE Press, Jun 2002.
- [95] S. BHOLA., Y. Z. and AUERBACH, J., “Scalably supporting durable subscriptions in a publish/subscribe system,” in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2003)*, pp. 57C66, June 2003.
- [96] SAITO, Y., KARAMANOLIS, C., KARLSSON, M., and MAHALINGAM., M., “Taming aggressive replication in the pangaea wide-area file system,” in *In 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 15–30, December 2002.
- [97] SHELDON, N., GIRARD, E., BORG, S., CLAYPOOL, M., and AGU, E., “he effect of latency on user performance in warcraft iii,” in *In Proceedings of the 2nd Workshop on Network and System Support for Games (NetGames 2003)*, pp. 3–14.
- [98] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., and BALAKRISHNAN, H., “Chord: A scalable peer-to-peer lookup service for internet applications,” *ACM SIGCOMM*, 2001.
- [99] SZYMANKIAK, M., PIERRE, G., and VAN STEEN, M., “Latency-driven replica placement,” in *Proceedings of the International Symposium on Applications and the Internet (SAINT)*, (Trento, Italy), Feburary 2005.
- [100] TANG, X. and XU, J., “On replica placement for QoS-aware content distribution,” in *INFOCOM*, 2004.
- [101] TOUCH, J., WANG, Y., EGGERT, L., and FINN, G., “A virtual internet architecture,” technical report, ISI, ISI-TR-2003-570, March 2003.
- [102] TRAVERSAT, B. and ET AL., “Project jxta 2.0 super-peer virtual network,” *Sun Microsystems, Inc.*, 2003.
- [103] VAN STEEN, M., HOMBURG, P., and TANENBAUM, A. S., “Globe: A wide-area distributed system,” *IEEE Concurrency*, vol. 7, pp. 70–78, January-March 1999.
- [104] W3C, *Simple Object Access Protocol (SOAP) 1.1*, 2000. URL: <http://www.w3c.org/TR/SOAP>.

- [105] WALDVOGEL, M. and RINALDI, R., “Efficient topology-aware overlay network,” *ACM Computer Communication Review*, vol. 33, pp. 101–106, Jan. 2003.
- [106] WATERS, B. and ANDERSON, D., “Locals and beacons: Efficient and precise support for large multi-uer virtual environments,” in *IEEE*, mar 1996.
- [107] WAUTERS, T., COPPENS, J., LAMBRECHT, T., DHOEDT, B., and DEMEESTER, P., “Distributed replica placement algorithms for peer-to-peer content distribution networks,” in *EUROMICRO*, pp. 181–188, 2003.
- [108] WESSELS, D., “Intelligent caching for world-wide web objects,” in *In Proc. INET95, Honolulu, Hawaii, June 1995. Internet Society*.
- [109] WOLLRATH, A., RIGGS, R., and WALDO, J., “A distributed object model for the Java system,” in *2nd Conference on Object-Oriented Technologies & Systems (COOTS)*, pp. 219–232, USENIX Association, 1996.
- [110] WONG, T., KATZ, R., and MCCANNE, S., “Evaluation of preference clustering in large-scale multicast applications,” in *inforcom*, 2000.
- [111] Y. CHAWATHE, S. M. and BREWER, E., “An architecture for internet content distribution as an infrastructure service,” *available at <http://www.cs.berkeley.edu/yatin/papers>*, 2000.
- [112] Y. CHU, S. R. and ZHANG, H., “A case for end system multicast,” in *Proceedings of SIGMETRICS 2000, pp. 1C12, 2000*.
- [113] YEE, W. G., OMIECINSKI, E., NAVETHE, S., AMMAR, M., DONAHOO, J., and MALIK, S., “A greedy approach for improving update processing in intermittently synchronized databases,” Tech. Rep. GIT-CC-99-18, Georgia Institute of Technology, jun 1999.
- [114] ZEGURA, E. W., CALVERT, K., and BHATTACHARJEE, S., “How to model an inter-network,” *IEEE Infocom*, 1996.
- [115] ZEGURA, E. W., CALVERT, K. L., and BHATTACHARJEE, S., “How to model an internetwork,” in *IEEE Infocom*, vol. 2, (San Francisco, CA), pp. 594–602, IEEE, March 1996.
- [116] ZHAO, B. Y., KUBIATOWICZ, J. D., and JOSEPH, A. D., “Tapestry: An infrastructure for fault-tolerant wide-area location and,” tech. rep., Berkeley, CA, USA, 2001.
- [117] ZHAO, Y., STURMAN, D., and BHOLA, S., “Scalably supporting durable subscriptions in a publish/subscribe system,” in *International Conference on Dependable Systems and Networks (DSN)*, 2003.
- [118] ZHAO, Y., STURMAN, D., and BHOLA, S., “Subscription propagation in highly-available publish/subscribe middleware,” in *ACM/IFIP/USENIX International Middleware*, 2004.
- [119] ZHOU, D., CHEN, Y., EISENHAEUER, G., and SCHWAN, K., “Active brokers and their runtime deployment in the echo/jecho distributed event systems,” in *AMS*, 2001.

- [120] ZHOU, D., CHEN, Y., EISENHAUER, G., and SCHWAN, K., “Jecho–interactive high performance computing with java event channels,” in *IPDPS*, 2001.
- [121] ZOU, L., AMMAR, M. H., and DIOT, C., “An evaluation of grouping techniques for state dissemination in networked multi-user games,” in *MASCOTS*, 2001.

VITA

Tianying Chang was born in Beijing, China. She received her Bachelor of Science degree in computer science in 1998 from Peking University, China. Tianying joined the Ph.D. program of College of Computing at the Georgia Institute of Technology in Fall 1999. Her research interests include distributed middleware system, peer-to-peer networks, overlay networks and multicast. When she is not busy with her research work, she enjoys traveling, playing tennis, skiing and playing accordion and piano.