

SCHEDULING FOR FAST TURNAROUND IN PEER-BASED DESKTOP GRID  
SYSTEMS

by

DAYI ZHOU

A DISSERTATION

Presented to the Department of Computer  
and Information Science  
and the Graduate School of the University of Oregon  
in partial fulfillment of the requirements  
for the degree of  
Doctor of Philosophy

June 2006



SCHEDULING FOR FAST TURNAROUND IN PEER-BASED DESKTOP GRID  
SYSTEMS

by

DAYI ZHOU

A DISSERTATION

Presented to the Department of Computer  
and Information Science  
and the Graduate School of the University of Oregon  
in partial fulfillment of the requirements  
for the degree of  
Doctor of Philosophy

June 2006

“Scheduling for Fast Turnaround in Peer-based Desktop Grid Systems,” a dissertation prepared by Dayi Zhou in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

*Virginia M. Lo*

\_\_\_\_\_  
Dr. Virginia Lo, Chair of the Examining Committee

*6/06/06*

\_\_\_\_\_  
Date

Committee in charge:           Dr. Virginia Lo, Chair  
  Dr. Allen Malony  
  Dr. Andrzej Proskurowski  
  Dr. Patricia A. Gwartney  
  Dr. Sharad Garg

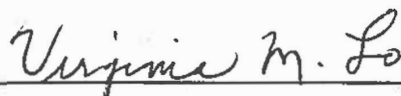
Accepted by:

*Tim King*

\_\_\_\_\_  
Dean of the Graduate School

An Abstract of the Dissertation of  
Dayi Zhou for the degree of Doctor of Philosophy  
in the Department of Computer and Information Science  
to be taken June 2006  
Title: SCHEDULING FOR FAST TURNAROUND  
IN PEER-BASED DESKTOP GRID SYSTEMS

Approved:



Dr. Virginia Lo, Chair

This dissertation focuses on *scheduling strategies* that achieve *fast turnaround* in open, dynamic, and large scale peer-based desktop grid. The challenges are two-fold: How does the scheduler quickly discover idle cycles in the absence of global information about host availability? And how can faster turnaround time be achieved within the opportunistic scheduling environment? Thus, our research is focused on two components of a peer-based desktop grid system: (1) fast scalable resource discovery and (2) application scheduling for fast turnaround.

We first describe a general peer-based desktop grid architecture, CCOF (Cluster Computing on the Fly). All of our research is based on this open, scalable, autonomous architecture.

Resource discovery is used to find available hosts in the peer-to-peer overlay network. We are the first to conduct a comprehensive study of generic resource discovery

methods in dynamic peer-based desktop grid systems. We found that the rendezvous point algorithm performs best under both light and heavy workloads because of its high success rate and consistently low message overhead.

We capitalized on the features of structured overlay networks to design two innovation rendezvous point selection schemes. SORPS (Structured Overlay Rendezvous Point Selection) selects a group of rendezvous points in a structured overlay network, with the goals of a balanced load among rendezvous points and low latency access from ordinary peers to rendezvous points. We then introduced *virtual rendezvous points*, which, in contrast to SORPS, do not utilize any physical nodes for resource discovery. Instead, we encode resource information in each node label, creating a *RAON (resource-aware overlay network)*.

We designed a new peer-to-peer infrastructure called WaveGrid which uses a timezone-aware RAON for fast resource discovery and migration from heavily loaded host to lightly loaded host to improve throughput. We evaluated the performance of WaveGrid using a heterogeneous host CPU profile based on statistical data derived from the BOINC volunteer computing system. The simulation results show that WaveGrid outperforms other systems with respect to turnaround, stability and minimal impacts on hosts.

This dissertation includes both my previously published and my co-authored materials.

## CURRICULUM VITA

NAME OF AUTHOR: Dayi Zhou

PLACE OF BIRTH: Xi'an, Shannxi, China

DATE OF BIRTH: November 17, 1976

## GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon  
Tsinghua University

## DEGREES AWARDED:

Doctor of Philosophy in Computer Science, 2006 , University of Oregon  
Master of Science in Computer Science, 2002, University of Oregon  
Bachelor of Engineering in Computer Engineering, 2000, Tsinghua University

## AREAS OF SPECIAL INTEREST:

Distributed Computing  
Peer-to-peer networks  
Grid computing

## PROFESSIONAL EXPERIENCE:

Teaching Assistant, Computer and Information Science Department,  
University of Oregon, Eugene, Oregon,  
2000 - 2006  
Research Assistant, Network Research Group, Computer and Information  
Science Department, University of Oregon,  
2000 - 2006

## PUBLICATIONS:

Dayi Zhou and Virginia Lo. WaveGrid: a Scalable Fast-turnaround Heterogeneous Peer-based Desktop Grid System, In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, April, 2006.

Virginia Lo, Dayi Zhou, Yuhong Liu, Chris GauthierDickey, and Jun Li. Scalable Supernode Selection in Peer-to-Peer Overlay Networks, In *Proceedings of International Workshop on Hot Topics in Peer-to-Peer Systems 2005 (HOT-P2P'05)*, July, 2005.

Dayi Zhou and Virginia Lo. Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-based Desktop Grid Systems, In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, June, 2005.

Xun Kang, Dayi Zhou, Dan Rao, Jun Li and Virginia Lo. Sequoia - A robust communication architecture for collaborative security monitoring Systems, In *SIGCOMM'04 Poster Session*, August, 2004.

Jun Li, Virginia Lo, Xun Kang, Dayi Zhou, and Dan Rao. Resilient and self-organizing overlay of collaborative security monitors, In *USENIX'04 Work-in-progress*, June, 2004.

Dayi Zhou and Virginia Lo. Cluster Computing on the Fly: Resource Discovery in a Cycle Sharing Peer-to-Peer System, In *Proceedings of the Fourth International Workshop on "Theory and Experience of Desktop Grids and P2P systems (GP2PC'04)*, April, 2004.

Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao. Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet, In *Proceedings of the Third International Workshop on Peer-to-peer Systems (IPTPS'04)*, February, 2004.

Daniel Zappala and Dayi Zhou. Performance Evaluation of Local Search Heuristics for QoS Multicast Routing, In *Proceedings of The 11th International Conference on Computer Communications and Networks (ICCCN'02)*, October, 2002.



## ACKNOWLEDGMENTS

I would like to first express my sincere appreciation to my advisor, Professor Virginia M. Lo, for her guidance and support during my graduate study. I am very lucky to have an advisor who is always willing to spend her time to listen to my problems no matter how small they might be. Without the long hours she spent and her valuable technical and editorial advice, this work could not have been completed. I thank Professor Lo for everything she has done for me.

I would also like to thank Professor Allen Malony, Professor Andrzej Proskurowski, Professor Patricia Gwartney and Dr. Sharad Garg for serving as my dissertation committee members and for their help during my thesis writing.

I thank the Department of Computer and Information Science for providing a friendly and supportive research environment and financial support during the years of my Ph.D study.

Finally, my parents receives my deepest gratitude and love for their dedication and sacrifice. I thank them for their support in all levels during my high school and undergraduate study, and their soothing encouragement when I encountered roadblocks in pursuing my goals.

To my late grandparents.

## TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION TO PEER-BASED DESKTOP GRID SYSTEMS . . .	1
1.1 Research Challenges . . . . .	5
1.2 Dissertation Roadmap . . . . .	7
2. BACKGROUND AND RELATED WORK . . . . .	11
2.1 Foundation Areas for Research in Peer-based Desktop Grid Systems	12
2.2 Peer-to-peer Networks . . . . .	13
2.2.1 Unstructured vs. Structured Overlay Networks . . . . .	14
2.3 Overlay Construction in Peer-to-peer Networks . . . . .	16
2.3.1 Overlay Construction in Unstructured Overlay Network . . . . .	17
2.3.2 Overlay Construction in Structured Overlay Network . . . . .	18
2.4 Resource Discovery in Peer-to-Peer Networks . . . . .	21
2.4.1 Resource Discovery in Unstructured Peer-to-peer Networks	22
2.4.2 Resource Discovery in Structured Peer-to-peer Networks . . . . .	22
2.5 Cycle Sharing Systems . . . . .	24
2.5.1 Institutional-based Load Sharing Systems . . . . .	25
2.5.2 Internet-based Computing Projects and Web-based Computing Projects . . . . .	27
2.5.3 Peer-based Desktop Grid Systems . . . . .	28
2.6 Resource Discovery and Application Scheduling in Peer-base Desktop Grid Systems . . . . .	30
2.6.1 Resource discovery in Peer-based Desktop Grid Systems . . . . .	30
2.6.2 Application Scheduling in Peer-based Desktop Grid Systems	33
3. A GENERAL ARCHITECTURE FOR PEER-BASED DESKTOP GRID SYSTEMS . . . . .	35
3.1 CCOF: a General Architecture for Peer-based Desktop Grid Systems	35
3.2 Suitable Applications for Peer-based Desktop Grid Systems . . . . .	39
3.3 Workpile Applications. . . . .	41

4. RESOURCE DISCOVERY IN PEER-BASED DESKTOP GRID SYSTEMS . . . . .	43
4.1 Introduction . . . . .	43
4.2 CCOF Resource Discovery Model . . . . .	44
4.2.1 Dynamic Hosts . . . . .	45
4.2.2 Profile Based Model . . . . .	46
4.2.3 Workpile Applications . . . . .	46
4.2.4 Search Algorithms . . . . .	46
4.2.5 Scheduling Strategies . . . . .	49
4.3 Simulation . . . . .	50
4.3.1 Simulation Configuration . . . . .	50
4.3.2 Simulation Results . . . . .	51
4.4 Conclusions . . . . .	58
5. DYNAMIC RENDEZVOUS POINT SELECTION IN PEER-TO-PEER OVERLAY NETWORKS . . . . .	61
5.1 Problem Definition and Related Work . . . . .	64
5.1.1 Criteria of the Rendezvous Points Selection Problem . . . . .	64
5.1.2 Related Theoretical Problems . . . . .	66
5.1.3 Rendezvous Point Selection in Unstructured Overlay Networks . . . . .	67
5.2 Rendezvous Points Selection in Structured Overlay Networks . . . . .	70
5.2.1 SORPS: Structured Overlay Rendezvous Point Selection . . . . .	70
5.2.2 Drawbacks of Physical Rendezvous Points . . . . .	77
5.3 Virtual Rendezvous Points Schemes . . . . .	78
5.3.1 Examples of RAON . . . . .	79
5.3.2 Advantages and Limitation of RAON . . . . .	81
5.3.3 Timezone aware RAON for Peer-based Desktop Grids . . . . .	82
5.3.4 Summary . . . . .	83
6. WAVEGRID: SCHEDULING FOR FAST TURNAROUND IN OPEN PEER-BASED DESKTOP GRID SYSTEMS . . . . .	86
6.1 Introduction . . . . .	86
6.2 Timezone-aware Overlay Network and Scheduling in WaveGrid . . . . .	89
6.3 Migration . . . . .	93
6.3.1 Suitable Applications for Migration in Peer-based Desktop Grid Systems . . . . .	94

6.3.2	Migration Strategies . . . . .	95
6.3.3	Peer-based scheduling strategies that utilize migration . . . . .	96
6.4	Simulation-based Evaluation of WaveGrid . . . . .	98
6.5	Heterogeneous Host CPU Power Profile . . . . .	99
6.6	Simulation configuration . . . . .	103
6.6.1	Basic Simulation Configuration . . . . .	103
6.6.2	Heterogeneous Environment and Rescheduling . . . . .	104
6.7	Evaluating Different Migration Strategies . . . . .	105
6.7.1	Simulation Metrics . . . . .	106
6.7.2	Simulation Results . . . . .	107
6.8	Simulation Experiments using a Heterogeneous CPU profile . . . . .	116
6.8.1	Simulation Metrics . . . . .	116
6.8.2	Simulation Results . . . . .	117
6.9	Extensions to WaveGrid . . . . .	123
6.10	Conclusion . . . . .	125
7. CONCLUSION AND FUTURE WORK . . . . .		128
7.1	Contribution . . . . .	128
7.2	Future Work . . . . .	132
7.2.1	Improvement to WaveGrid . . . . .	132
7.2.2	Peer-to-peer Checkpointing . . . . .	133
7.2.3	Fairness in Peer-based Desktop Grid Systems . . . . .	134
BIBLIOGRAPHY . . . . .		136

## LIST OF FIGURES

Figure	Page
1.1 The different architectures of client-server model and peer-to-peer model .	4
2.1 Node join in CAN . . . . .	19
2.2 Resource discovery in CAN . . . . .	23
2.3 Relationship between peer-based cycle sharing systems and their ancestor systems . . . . .	25
3.1 CCOF Architecture . . . . .	36
3.2 Client submits job to CCOF. . . . .	37
3.3 Host receives request from client. . . . .	37
4.1 Job completion rate under uniform workload, when the ratio of clients to donors is 0.1. . . . .	52
4.2 Message overhead (for 24-hour period) under uniform workload, when the ratio of clients to donors is 0.1. . . . .	53
4.3 Average distance from clients to donors under uniform workload, when the ratio of clients to donors is 0.1 . . . . .	54
4.4 Job first submission success rate under uniform workload, when the ratio of clients to donors is 0.7 . . . . .	55
4.5 Job completion rate under uniform workload, when the ratio of clients to donors is 0.7. . . . .	56
4.6 Job migration failure rate under uniform workload, when the ratio of clients to donors is 0.7 . . . . .	57
4.7 Job completion rate under normal workload, when the ratio of clients to donors is 0.7. . . . .	58
4.8 Message overhead under uniform workload, when the ratio of clients to donors is 0.7 . . . . .	59
5.1 Gnutella two-tier hierarchical overlay network. . . . .	68
5.2 Rendezvous point selection in CAN. (Rendezvous point label expression is $(0.05+0.25 n)\%1,(0.20+0.25 m)\%1$ . . . . .	72
5.3 An example of RAON, the information of operating system and CPU clock rate on hosts is embedded into the overlay network. . . . .	80
5.4 An example of Timezone-aware RAON. Hosts join according to their time-zone information. . . . .	83
6.1 Job initiation and migration in WaveGrid . . . . .	91
6.2 Sample host profile of available idle cycles . . . . .	93

6.3	CPU powers of different CPU groups using empirical data from BOINC statistic (collected in Aug. 2005). The credits are averaged over all the CPUs in the same group. . . . .	101
6.4	Percentage of different CPU groups with different ranks . . . . .	102
6.5	Average slowdown factor for no-migration vs. migration (The percentage of clients in the system is 20%) . . . . .	106
6.6	histogram of slowdown factors of successfully finished jobs (The percentage of clients is 20% and the percentage of free time on hosts is 15%) . . . . .	109
6.7	% of jobs that fail to complete(The percentage of clients in the system is 20%) . . . . .	110
6.8	% of jobs that fail to complete (The percentage of free time on the hosts during the day is 15%) . . . . .	111
6.9	Average number of migrations for successfully finished jobs (The percentage of clients in the system is 20%) . . . . .	112
6.10	Wave Scheduler: Average slow down factor(The percentage of client request is 20%) . . . . .	113
6.11	% of job that fail to complete (The percentage of clients in the system is 20%) . . . . .	114
6.12	% of job that fail to complete (The percentage of available time on the hosts during the day is 15%) . . . . .	115
6.13	Average number of migrations (The percentage of client in the system is 20%) . . . . .	116
6.14	Average makespan vs host availability (The percentage of clients is 20%)	118
6.15	Average slowdown vs host availability (Percentage of clients is 20%) . .	119
6.16	Histogram of distribution of makespan (Percentage of clients is 20%. Percentage of available time on hosts during the day is 40%.) . . . . .	120
6.17	Average number of migrations (Percentage of clients is 20%) . . . . .	121
6.18	Average makespan when the migration delay varies (Percentage of clients in the system is 20%. Percentage of free time on hosts during the day is 50%.) . . . . .	122
6.19	Average number of retries during the job execution (Percentage of clients is 20%) . . . . .	123
6.20	Average slowdown vs host availability when using random host selection (Percentage of clients is 20%) . . . . .	124
6.21	Percentage of performance improvement when using the most powerful host selection instead of a random host selection (Percentage of clients is 20%) . . . . .	125
6.22	Average slowdown when using the eager migration strategy (Percentage of clients is 20%) . . . . .	126
6.23	Percentage of performance improvement when using eager migration (Percentage of clients is 20%) . . . . .	127

## LIST OF TABLES

Table	Page
2.1 Comparison of unstructured to structured overlay networks . . . . .	15
2.2 Properties of different structured overlays (n is total number of peers. d is the number of dimensions in CAN.) . . . . .	19
2.3 Existing Peer-based Cycle sharing Systems . . . . .	29
3.1 Examples of workpile applications . . . . .	42
4.1 Average/Max size of jobs scheduled under uniform workload, when the ratio of clients to donors is 0.7. (Job size is measured in unit of processor-hours) . . . . .	56
5.1 Comparison of SORPS with rendezvous point selection in unstructured overlay network (RP: rendezvous point) . . . . .	84
5.2 Comparison of physical rendezvous point with virtual rendezvous points (RP:rendezvous point) . . . . .	84
6.1 Different migration strategies . . . . .	97
6.2 Sample statistical data for SETI@home organized by types of CPUs, which groups hosts according to type of its CPU. (Note: "Average credit" is the average credit granted over the last few days to the hosts with this type of CPU.) . . . . .	100



# CHAPTER 1

## Introduction to Peer-based Desktop Grid Systems

Peer-based desktop grid systems represents a new research area stimulated by the recent success of Internet-based computing projects such as SETI@home and BOINC and rapid developments in peer-to-peer computing. The goal of peer-based grid systems is to support the automatic execution of CPU intensive i.e. long-running applications in a network of voluntary peers willing to donate idle cycles. These systems aggregate idle cycles from host machines on the edge of the Internet into a large resource pool and manage these resources in an efficient, fair, and secure way. here are two types of nodes in a cycle sharing system: the *client* and the *host*. The host donates idle cycles and the client seeks extra cycles for its tasks. In a peer-to-peer cycle sharing system, a node can be either a client or a host or both. Research in peer-based desktop grid systems lies in the intersection of distributed computing research and peer-to-peer networking research, and represents a new generation of cycle sharing systems.

It has long been acknowledged that a vast amount of cycles lie idle on user machines, and the advantage of utilizing idle cycles through automatic scheduling tools has not gone unnoticed. A number of predecessors of cycle sharing systems have paved the way for research in peer-based desktop grid systems. These include load sharing systems, Internet-based computing projects and scientific Grids.

In the mid 1980's, load sharing in distributed systems emerged as an approach to utilize idle cycles within one institution. Condor [63, 94], which was implemented at University of Wisconsin, is one of the most widely recognized load sharing projects. The original Condor system was built as a production system within one institution using a central matchmaking server for resource management. With the development of new techniques, Condor evolved to include more than one institution. Condor-Flock [33] proposed to build a Condor pool of multiple institutions via connecting matchmaking servers at different sites. CondorG [94] advanced the research by utilizing Grid computing techniques to connect multiple sites. However, it is hard to expand Condor to a system without institutional support, to allow machines not associated with any institutions to join. The central servers may also become performance bottlenecks and points of failure in a large scale system.

Internet-based computing projects were built about the same time as Condor. They focus on utilizing idle cycles scattered throughout the Internet. The most noticeable design features of these systems are a client-server architecture, rigid master-slave scheduling model, and strict protection of local jobs' privileges. In such a system, the volunteer hosts have to manually download and install hosting software. The central sever then dispatches jobs to the registered hosts. The foreign jobs run on the hosts under a screensaver model; that is, they run when there are no recent mouse or keyboard activities, otherwise they sleep. The Internet-based computing projects have attracted a large group of users, which shows the potential for utilizing idle cycles on the Internet. For example, BOINC has over 500,000 registered users. On the other hand, the frequent down time of servers in these systems suggests there is need for a more scalable infrastructure than the client-server architecture.

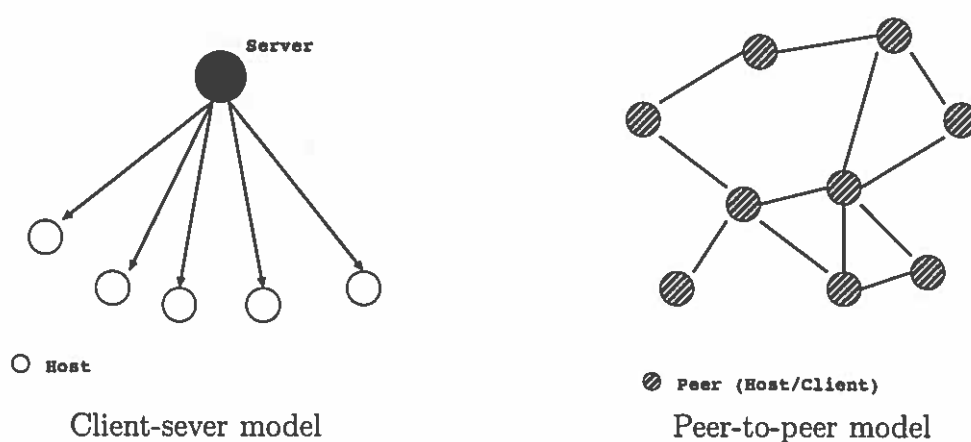
Finally, traditional Grid systems connect federated computing sites such as supercomputer labs and then provide tools and services to users within the community [38, 8]. Traditional grid computing infrastructures use matchmaking schemes that are similar to those used in the distributed systems [65] or specialized protocols [27] for negotiating resource usage and to locate available resource. Therefore, they are costly to assemble and operate.

The most recent type of cycle sharing systems are known as desktop grid systems. Desktop grid systems are low-end grids which harness idle cycles on desktop machines at the edge of the Internet. These systems can serve the needs of a wide user population, ranging from research scientists running compute-intensive scientific applications to commercial enterprises specializing in multimedia and digital arts processing, to high school students engaged in science or art fair projects. Our work and that of systems including flock of condors [12], OurGrid [3], CompuP2P [45] and PersonalGrid [46] advances the state of the art by utilizing the newest techniques from peer-to-peer networking in the design of desktop grid systems.

A *light-weight, open peer-based desktop grid system* is a desktop grid system built using peer-to-peer techniques and is thus easy to assemble and operate. It allows cycle donors with similar interests to self-organize into a cycle sharing community by joining an *overlay network*, similar to those used by peer-to-peer file sharing systems. In contrast to the institutional-based Grid systems [37, 94], load sharing systems in local networks [63, 106], and client-server-based Internet computing projects [2, 75, 90, 6], peer-based desktop grid systems circumvent the performance bottleneck of central servers and are capable of achieving much larger scale. Moreover, a peer-based desktop grid system allows each peer to be a potential donor of idle cycles as well as a potential source of tasks for automatic scheduling in the virtual resource pool. The peer-based model is distinctly different from the client-server-based global computing projects such as BOINC, SETI@home and Stanford Folding. In these systems all the volunteer machines donate idle cycles to a single scientific application but reap no benefits for their own computational needs. Figure 1.1 contrasts the client-server model and the peer-based cycle sharing model.

Although peer-based desktop grid systems are scalable, lightweight, and easy to assemble and operate, many open problems remain to achieve the highest quality of service possible in an dynamic large-scale desktop grid environment such as fast turnaround scheduling, fault-tolerance and security.

The problem addressed in this dissertation is the development of *scheduling strategies* that achieve *fast turnaround time* in open, dynamic, and large scale peer-based desktop grid systems. Scheduling is the essential function of a peer-based desktop



**FIGURE 1.1:** The different architectures of client-sever model and peer-to-peer model

grid system and key to the performance of such systems. While scheduling for fast turnaround has always been the goal of traditional scheduling research, compared with scheduling in a dedicated environment, scheduling in an opportunistic cycle sharing environment is more challenging. Scheduling in a peer-based cycle sharing system is even more changing as the hosts are connected by a dynamically changing virtual overlay network and the size of the system is potentially very large.

The dissertation is the first to address scheduling for fast turnaround in a large scale desktop grid system. The scheduling methods in traditional cycle sharing systems are not suitable in this new cycle sharing infrastructure. Condor style scheduling requires central servers for resource discovery and scheduling, and therefore cannot be transplanted into an Internet-scale system beyond the boundaries of institutions. Existing Internet-scale cycle sharing systems use best effort scheduling techniques relying entirely on external incentives such as fame. These systems also suffer from bottlenecks due to client-server architecture. Present research [57, 58] in scheduling for fast-turnaround in desktop grid systems assumes the use of central servers; therefore these results cannot be used in a peer-based system.

## 1.1 Research Challenges

The design of scheduling strategies to satisfy higher quality-of-service (QoS) requirements, such as fast turnaround and high throughput, is a big challenge in dynamic, opportunistic peer-based cycle sharing systems, which faces a number of unique challenges inherent to the nature of the peer-to-peer environment.

The challenges first come from the opportunistic and volatile nature of the peer-based cycle sharing systems. Peer-based cycle sharing systems use non-dedicated machines in which local jobs have much higher priority than foreign jobs. Therefore, compared to running on dedicated machines, the foreign job will make slower progress since it can only access a fraction of the host's CPU availability. The resources are highly volatile in a peer-based cycle sharing system. Nodes may leave and join the system at any time, and resource owners may withdraw their resources at any time. Therefore, the foreign jobs may experience frequent failures due to the volatility of the resources.

The challenges for design of an efficient scheduling system for peer-based desktop grid systems also come from the difficulties in collecting global and accurate resource information. It is unscalable to collect resource information such as idle time slots, CPU speed and type of operating system for all the nodes in a large scale peer-based cycle sharing system. Also users of peer-based cycle sharing systems may find it is intrusive to report their computer usage information periodically to some unknown remote clients. Therefore, scheduling in large scale cycle sharing systems are usually best effort scheduling based on limited resource information.

The problem of scheduling in peer-based cycle sharing systems encompasses all the activities involved with utilizing idle cycles in a distributed, open environment: overlay management, resource discovery, application scheduling, local scheduling on the host, and meta-level scheduling which involves coordination of efforts among a community of application-based schedulers. Our research is focused on two of these problems: (1) fast scalable resource discovery methods and (2) application scheduling for fast turnaround.

- **Resource discovery.** The goal of resource discovery in peer-based cycle sharing systems is to discover available hosts in the ocean of hosts, even when hosts leave and join the system dynamically, and the amount of resources dynamically change. This dissertation addresses fundamental questions in the area of resource management within large scale, dynamic, opportunistic computing environments: what is the best scalable protocol for searching through a large network of candidate nodes to discover those qualified and willing to donate cycles? How can resource information best be represented and kept up to date in a dynamically changing environment?
- **Application scheduling.** The goal of application scheduling is to choose the best hosts on which to schedule the job, even when the resource information is limited, imprecise, and inaccurate. There are many open questions in this field. How can the scheduler adjust quickly to dynamic changes in the availability of the machine hosting the foreign code? Should the task continue to reside on the current host machine or migrate to a new machine with potential for better service? Can a peer-based cycle sharing system meet reasonable QoS requirements regarding turnaround time to complete the task(s)?

Scheduling in peer-based cycle sharing environment represents a new body of research resulting in the synergy of both distributed computing and networking, which pushes the frontier far beyond scheduling in a dedicated environment or within one or several institutions. Our work is part of the cross-fertilization of ideas and discoveries in this evolving research domain. As an immediate result, our work will enhance the computing environment for a diverse population of users and their applications, ranging from scientists to ordinary users. In the long run, our work contributes valuable guidelines to design of scheduling strategies in a wide range of computing environments, spanning the spectrum from open, ad hoc communities of low-end clients, to federated Grid computing systems.

## 1.2 Dissertation Roadmap

Starting with the design of the general architecture, we first focused on resource discovery methods in this type of system and schemes to aid resource discovery such as scalable rendezvous point selection. Based on results of these study, we designed the scheduling methods for fast turnaround in this open, fully distributed and heterogeneous peer-based infrastructure. This dissertation work can be divided into four phases:

- **General peer-based cycle sharing architecture.** We first designed a general peer-based cycle sharing architecture, CCOF (Cluster Computing on the Fly). All of our research is based on this open, scalable, autonomous architecture, and follows the same light-weight and non-intrusive design philosophy. CCOF is composed of a community-based overlay network, application scheduler, local scheduler and coordinated scheduler. In the CCOF architecture, hosts join a variety of community-based overlay networks, depending on how they would like to donate their idle cycles. Clients then form a compute cluster on the fly by discovering and scheduling on sets of machines from these overlays. CCOF charts many research challenges, including incentives and fairness, trust and reputation, security and performance monitoring. At this stage, we also identified four types of applications, which are suitable to run in this type of systems: infinite workpile application (bag-of-task), workpile applications with deadlines, tree-based search applications, and point-of-presence applications. Among these applications, We focus on the scheduling of workpile applications, those that consume a large amount of cycles with little or no data communication.
- **Resource discovery methods.** Resource discovery is used to collect host information and to find available hosts in the peer-to-peer overlay network. The focus is to understand what scalable generic resource discovery method is the best for fast turnaround cycle sharing. Though resource discovery was widely studied in peer-to-peer file sharing system, it was unknown whether the results

can be directly applied to a cycle sharing system, as cycles and data are inherently different types of resources. We are the first to conduct a comprehensive study of four generic resource discovery methods in a dynamic peer-based cycle sharing environment: expanding ring search, random walk, advertisement-based search, and rendezvous points based search. For comparison purposes, a centralized search algorithm was also included in the study, which is optimal but not scalable. We found that the rendezvous point algorithm performs best overall under both light and heavy workloads because of its consistently low message traffic overhead.

- **Rendezvous point selection methods.** Rendezvous point selection methods in peer-based desktop grid systems are used to select a group of peers to function as matchmaking servers. Hosts submit their resource profiles to rendezvous points, and rendezvous points match client requests with suitable hosts. Based on our study about generic resource discovery methods which shows that rendezvous point based resource discovery performs the best, We tackled the open problem of placing a dynamic group of rendezvous points in a large scale overlay network. The goals for placement are a balanced load among rendezvous points and low latency access from ordinary peers to rendezvous points. Existing rendezvous point selection methods in unstructured overlay networks cannot evenly distribute the rendezvous point throughout the overlay networks. In addition, they all rely on probing with high message overhead to discover rendezvous points.

I capitalized on the features of structured overlay network to design two innovative rendezvous point selection schemes. The first rendezvous point selection algorithm, SORPS (Structured Overlay Rendezvous Point Selection) selects a group of rendezvous points in a structured overlay network, with a goal of keeping the rendezvous point to ordinary peer ratio stable as peers join and leave the overlay. Although SORPS is scalable and can achieve access and load balance requirements, it relies on physical nodes to serve as rendezvous points and thus require complex maintenance and is susceptible to malicious nodes' attack.



Therefore, we introduced the notion of *virtual rendezvous point* which does not puts burdens onto one or a few physical rendezvous points. Instead, a special node labeling scheme categorizes hosts according to their resources to create virtual rendezvous point in the node label space. Hosts register themselves with the virtual rendezvous points according to their resource profiles using the node label formula, and the clients' requests are automatically routed to matching hosts with low resource discovery overhead.

- **Migration-based scheduling for fast-turnaround.** A host may withdraw cycles or completely leave from the peer-based desktop grid system, and then the foreign jobs cannot continue execution on that host. This is the major reason for the relatively long computation time in peer-based desktop grid systems compared with computation time in a dedicated machine. Migration is a technique used in distributed systems to migrate jobs from heavily loaded hosts to lightly loaded hosts to improve throughput. We are the first to apply this technique to peer-based cycle sharing environments to provide continuous execution. We answered the challenge of selecting the best migration target and migration time in this large dynamic system by using the virtual rendezvous point idea for fast resource discovery and to provide a suitable migration target.
- **WaveGrid: timezone-aware overlay network** Combining the concept of virtual rendezvous points and fast migration, we designed a new peer-to-peer scheduling infrastructure called WaveGrid. WaveGrid integrates a novel timezone-aware overlay construction method with task migration. Hosts organize themselves into a timezone-aware structured overlay network, which supports fast and straight-forward resource discovery. Hosts with potentially long chunks of idle night cycles will be identified as candidates for scheduling or as migration targets. Scheduling methods in WaveGrid takes heterogeneity into account in choosing hosts. WaveGrid then rides the wave of available cycles by migrating jobs to hosts located in night-time zones around the globe. We evaluated the performance of WaveGrid using a heterogeneous host CPU profile based on statistical data derived from the BOINC volunteer computing system.

The rest of the dissertation is organized as following. Chapter 2 summarizes the previous cycle sharing systems in distributed computing and the related research in peer-to-peer networks, especially overlay construction and resource discovery. Chapter 2 also surveys existing peer-based desktop grid systems discussing their different research goals, resource discovery schemes and scheduling methods. Chapter 3 describes describes general peer-based cycle sharing architecture, **CCOF** (Cluster Computing on the Fly). Chapter 4 reports our performance study of generic resource discovery methods in peer-based desktop grid systems. Chapter 5 describes Structured Overlay Rendezvous Point Selection (SORPS) and the concept of virtual rendezvous points. Chapter 6 describes **WaveGrid**, a scalable, heterogeneous peer-based desktop grid system for fast turnaround utilizing virtual rendezvous points and migration. Chapter 7 concludes the dissertation with key results and observation about fast resource discovery and scheduling for fast turnaround in peer-based desktop grid systems. The last part of dissertation describes future research in advanced features of WaveGrid, comprehensive host resource profiling and fault-tolerance schemes to supplement fast turnaround scheduling.

## CHAPTER 2

# Background and Related Work

Peer-based desktop grid systems is a new area of peer-to-peer networking research whose goal is to support the execution of user applications in a network of voluntary peers willing to donate idle cycles. It aggregates the idle cycles from the host machines on the edge of the Internet into a large resource pool and manages these resources in an efficient, fair, and secure way. Peer-based desktop grid systems inherit many advantageous design philosophies and techniques from the previous cycle sharing systems. More importantly, they adapt new peer-to-peer techniques, thus have many advantages over the traditional cycle sharing systems, such as scalability, light-weight, and easy to assembly and operate.

The idea of capturing idle cycles on desktop workstations first arose in the mid 1980's in the environment of local area distributed systems. Distributed institution-based load sharing systems were developed to automatically schedule and execute user applications using idle cycles within one or several institutions. One of the most popular load sharing systems, Condor [63, 95, 33, 41], was implemented in 1982 to serve as major source of computing cycles to faculty and students at the University of Wisconsin. The original Condor system utilized idle cycles within one institution, using a centralized server to discover and manage available resources.

About the same time as Condor, Internet-based computing projects [2, 84, 75, 29, 49] on the Internet emerged. Internet-based computing projects projects harness idle cycles from volunteering users to serve one particular application, while users recap

no benefits for their own applications. The project SETI@home [84] (Search for Extraterrestrial Intelligence) which began in 1982, uses voluntary machines to analyze signals from outer space for artificial radio signals coming from stars. SETI@home requires clients to manually download specialized software from the SETI@home server. The process running on the client side will report results to the same server. Today, SETI@home has over 5 million concurrent users who donate cycles.

The peer-based desktop grid systems [11, 12, 42, 67, 3, 46] combine the advantage of previous cycle sharing systems such as institutional based load sharing systems and Internet-based computing projects, while overcoming their shortcomings. Peer-based desktop grid systems eliminate the performance bottleneck of the central servers found in both institutional-based load sharing systems and Internet-computing projects, as the responsibility of resource discovery and scheduling are distributed among cooperated nodes in peer-to-peer overlay networks. Like SETI@home, peer-based desktop grid systems harness cycles from ordinary users on the edge of the Internet and can potentially be of very large scale. However, Peer-based desktop systems are more general than systems like SETI@home because they are not limited to a single application but support execution of arbitrary client applications. Like Condor's load sharing system, peer-based desktop grid systems support automatic resource discovery, scheduling and execution of user applications. However peer-based desktop grid systems are much more open, allowing average citizens to participate and not requiring a central server to be configured nor any institutional support.

## 2.1 Foundation Areas for Research in Peer-based Desktop Grid Systems

Peer-based desktop grid systems lie in the conjunction of two major fields of computer science research: cycle sharing systems and peer-to-peer networks. Thus, research in this field requires a deep understanding of principles and methods from both of the fields.

The advantage of using idle cycles is well-received and explored by many cycle sharing systems. Load sharing tools for traditional distributed systems, such as Condor [63], pool idle cycles within one or more participating institutions. Internet-based computing projects uses a client-server model to recruit volunteering hosts in supporting one particular long-running scientific application. With the emergence of peer-to-peer techniques, new peer-based cycle sharing architectures are proposed. Peer-based desktop grid systems are built in belief of the convergence of peer-to-peer networks and the Grid computing predicted in several studies [39, 61, 40, 36]. This chapter briefly surveys the cycle sharing systems including institutional based load sharing systems, Internet-based computing projects, and peer-based cycle sharing systems.

Peer-based desktop grid systems use peer-to-peer techniques evolved from peer-to-peer file and information sharing systems, ranging from popular file sharing technologies such as Gnutella to sophisticated information sharing and distributed hash table (DHT) systems. All these peer-to-peer systems presume a large-scale, open, insecure, and untrusted environment. Peer-to-peer techniques especially the overlay management are adopted in peer-based desktop grid systems. This chapter introduce the peer-to-peer overlay construction protocols and resource discovery in peer-to-peer networks, which are foundations of peer-based desktop grid systems.

## 2.2 Peer-to-peer Networks

Peer-to-peer networks emerged in 1999 with the popular file-sharing peer-to-peer system Napster. Soon a wealth of academic research developed to investigate the structure, scalability, and performance of peer-to-peer networks. In contrast to traditional client-server systems in which servers, often the more powerful machines, are dedicated to serving the clients, a peer-to-peer network is one in which functions of the systems are distributed among all the nodes, and any node can provide services to others. In a word, in a peer-to-peer network, nodes can be function as both server and client.

Peer-based desktop grid systems directly inherit many of the properties and characteristics of peer-to-peer networks in general. It organizes large number of machines on the edge of the Internet into an overlay network for sharing resources. Like peer-to-peer information sharing systems, it does this in a dynamic, open, unreliable, and untrusted environment. Therefore, the research achievements from peer-to-peer networks are highly relevant to the research in peer-to-peer cycle sharing systems, especially that addressing fundamental problems such as overlay construction, resource discovery, incentives, fairness, fault tolerance, security, and trust.

This section will present results from two key fields in peer-to-peer networks: overlay construction and resource discovery, which are most relevant to my research. Other issues such as incentives and fairness, fault tolerance, and most security-related issues are outside the scope of this dissertation.

### 2.2.1 Unstructured vs. Structured Overlay Networks

An *overlay network* is a virtual network that is overlaid on top of the physical network of the Internet. It allows peers to communicate directly with each other at the application level, making the details of message routing through the Internet invisible to the application. Peers who are connected to each other in the overlay network are referred to as neighbors and communication in an overlay network uses neighbor-to-neighbor communication, i.e. hop-by-hop message passing in the virtual overlay network.

Current peer-to-peer systems can be divided into two classes based on the characteristics of the underlying network overlay topology used for connecting the peers: *unstructured overlay networks* (also called random-mesh topologies) and *structured overlay networks*. Unstructured peer-to-peer overlay networks were the approach taken by the popular peer-to-peer applications such as Gnutella [56] and BitTorrent [24]. Unstructured overlay networks, as their name implies, do not presume any structure. Peers simply connect to other peers in an unconstrained manner forming an unstructured *mesh*. Thus, they are easier to build but require more effort for

**TABLE 2.1:** Comparison of unstructured to structured overlay networks

	Topology	Diameter	Number of Neighbors	Routing	Shared Storage	Data Migration When Node Leave
<b>Unstructured Overlay Networks</b>	Random Mesh	No explicit bound	Local policy, demonstrate similarity to a small world topology	Flooding based	Not required	Not required
<b>Structured Overlay Networks</b>	Regular topology: such as Ring, n-torus, butterfly	Within theoretical upper-bound	Uniform with a theoretical bound	Node label-based explicit routing	Required	Required

resource discovery. They are well-suited to best effort applications which are tolerant of the longer latencies incurred by flooding-based search through the unstructured overlay.

Structured overlay networks were originally designed for indexed lookup in a distributed file system. By imposing a regular or mathematical node labeling scheme onto the nodes of the overlay network, peers are able to efficiently look up information using an index (such as a hash of the file names) using node label-based routing. Several research projects on structured overlay networks have been highly visible, including CAN [78], Chord citeChord, Tapestry [102], and Pastry [83] which provide a basic distributed hash table (DHT) model for information lookup. A rich array of applications were built on the structured DHT overlays. To name a few, OceanStore [80] which is a data storage trading system built on Tapestry; Scribe [19] which is an application layer multicast protocol built on Pastry, and Squirrel [54] which is a web caching service also built on Pastry.

Table 2.1 compares unstructured overlay networks with structured overlay networks. The fundamental difference between the two lies in the different types of topology used, different node labeling schemes and different content storage schemes. The structured overlay networks take advantage of the power of regular networks: symmetric topology with bounded diameter and efficient node label-based routing schemes, so they can reduce the message overhead in locating contents. In addition, routing in an unstructured overlay network is not as efficient as in a structured overlay

network. The structured overlay network uses regular node labels, which corresponds to locations in the overlay network. Therefore, It can use an explicit label-based routing, which requires minimum route computation time. In contrast, the unstructured overlay network does not use a regular node labeling scheme. As a result, it does not and cannot support direct communication with other peers in the overlay network, but relies on broadcast to discover peers.

However a structured overlay is much more complicated to implement and hard to maintain. The structured networks requires that each peer in the system store some index information (keys) for the contents shared in peer-to-peer networks. While the storage requirements may not be excessive on the peers, since the keys are small, the maintenance of such keys in the system is a bigger issue. Keys need to be added and deleted from the overlay, when the file is created or deleted. When a node leaves the system, the keys owned by it need to be properly transferred. To handle node failure, keys often need to be replicated on several peers. The unstructured overlay network does not required shared storage to store globally consistent indexes of file. Some unstructured networks may require caching local file indexes on the peers functioning as rendezvous points, but they do not require storing any state on the ordinary nodes and maintain a globally consistent state. The maintenance problem of structured overlay network becomes an even harder problem under *churn*, which is the phenomenon of node continuously frequent arrival and departure. Many papers have proposed to address churn for structured overlay network [81, 86, 1].

## 2.3 Overlay Construction in Peer-to-peer Networks

The primary requirement for peer-to-peer system is scalability with the ever-increasing population of the peers. A system with centralized server, such as the now defunct Napster system, is not scalable. To organize millions of nodes, scalable self-organized peer-to-peer overlay construction methods are required.



### 2.3.1 Overlay Construction in Unstructured Overlay Network

An unstructured overlay construction protocol, namely Gnutella [23], was proposed, after Napster. An unstructured overlay network builds a random topology and it is simple. The original Gnutella [23] protocol builds a flat overlay network. When the node wants to join the system, it connects to the overlay network by establishing a TCP connection with some existing peer (the predominating way to acquire address of another peer is via address caching). The peer may flood a Ping message in a limited area to discover more neighbors, once it is in the system. On receiving such Ping message, peers can choose to respond with Pong message which contains the address of a peer and the amount of data it shares.

The flooding based key word search causes high network traffic overhead, which is a serious problem especially when involving peers with slow dial-up connections. The latest Gnutella protocol [56] builds a two-level hierarchical overlay to solve this problem. *Ultrapeers*, which are powerful nodes with high-speed network connections in the system, form the core of the overlay, while the other non-ultra peers (so called *leaf* peers) are attached to the core by connecting to some ultrapeers. Ultrapeers function as proxies for leaf nodes (cache indexes of files shared by leaf nodes and forward queries for leaf nodes). Before a new node joins the system, it is up to the node itself to pre-decide whether it can potentially be an ultrapeer or a leaf peer according to the following metrics: suitable operating system, sufficient bandwidth, sufficient uptime and sufficient RAM and CPU speed. The new node joins the overlay network according to basic Gnutella protocol described in the previous paragraph, but Ultrapeer capabilities and information are exchanged during the join procedure to select Ultrapeers. The new node can then connect to an Ultrapeer if it is not qualified to be an untrapeer, or it promotes itself to be an ultrapeers and let existing leaf node to connect to it.

Currently, the unstructured overlays with ultrapeer or supernodes are used in the most popular peer-to-peer file sharing systems. There are several other file sharing systems similar to Gnutella developed after Gnutella, such as KazaA [85] which uses supernodes similar to the ultrapeer in Gnutella. In such systems, the probing procedure is more efficient with less flooding activities (ultrapeers only forward the queries to leaf nodes when it is possible that the leaf nodes have the related files) and smaller flooding scope (flooding is mostly among ultrapeers). However, the new overlay network construction method does not change the flooding nature of the original protocols, which is rooted in lack of file location information.

In addition to the unscalable flooding based schemes, the Gnutella style overlay network construction protocol has other drawbacks, such as non-topology-sensitive, since the overlay network built by such a protocol may not map well to the underlying physical network. New methods are proposed to improve the topological sensitivity. Liu et.al. [64] suggests improving the performance of the Gnutella network by trimming the low performance redundant links over time, however this local adjustment may not finally achieve a global optimized state.

### 2.3.2 Overlay Construction in Structured Overlay Network

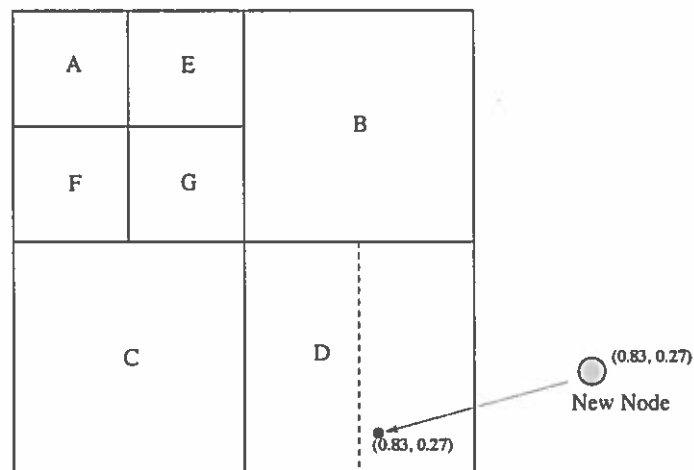
Structured overlay networks were developed to overcome some of the drawbacks of unstructured overlay networks. They exploit the power of regular topologies to reduce the resource discovery overhead. The structured overlays provide symmetric and balanced topologies, explicit labelbased or Cartesian distance based routing and theoretical-bounded virtual routing latency and neighbor table size. The most well-known structured overlays, CAN [34], Chord [35], Pastry [36] and Tapestry [37] use an n-torus, hypercube, and prefix-based ring respectively. Consistent hash methods motivated by the distributed hash algorithms are used to store and locate content, thus, this types of overlay networks are also called DHT (Distributed Hash Table)

**TABLE 2.2:** Properties of different structured overlays ( $n$  is total number of peers.  $d$  is the number of dimensions in CAN.)

	Topology	Routing	Neighbor	Shortest Path
<b>Chord</b>	Ring	Numeric Node label based	$O(\log n)$	$O(\log n)$
<b>CAN</b>	n-torus like	Cartesian coordinate based	$O(d)$	$O(n^{1/d})$
<b>Pastry</b>	Ring	Pre-fixed based routing	$O(\log n)$	$O(\log n)$
<b>Tapestry</b>	Ring	Pre-fixed based routing	$O(\log n)$	$O(\log n)$

overlay network. Table 2.2 compares difference between different structured overlays in regarding to the topology used, routing method, size of the neighbor (routing) table and length of the shortest path.

In a structured overlay network, the node ID of the peer decides its location in the overlay network, as well as its neighbors. When a node joins the system, it first generates a virtual node ID (by either hashing the IP address or the public key, or by picking a coordinate in the Cartesian space). Then a join message is routed toward the location corresponding to that node ID. As an example, we will explain the CAN overlay construction in details.



**FIGURE 2.1:** Node join in CAN

The CAN structured overlay [78] uses a Cartesian coordinate space. The entire space is partitioned among all the physical nodes in the system at any time, and each physical node owns a distinct subspace in the overall space. Any coordinate that lies within its subspace can be used as an address to the physical node which owns the subspace. For example, in Figure 2.1, the whole Cartesian space is partitioned among 7 nodes, A,B,C,D,E,F and G. The neighbors of a given node in the CAN overlay are those nodes who are adjacent along  $d - 1$  dimensions and abut along one dimension. The neighbors of node G are nodes F, E, B, and C.

In Figure 2.1, new node N joins the CAN space by picking a coordinate in the Cartesian space and sending a message into the CAN destined for that coordinate. (The method for picking the coordinate is application-specific.) There is a bootstrap mechanism for injecting the message into the CAN space at a starting node. The message is then routed from node to node through the CAN space until it reaches node D who owns the subspace containing N's coordinate. Each node along the way forwards the message to the neighbor that is closest in the Cartesian space to the destination. When the message reaches D, it then splits the CAN space with the new node N and adjusts the neighbor tables accordingly. Latency for the join operation is bounded by  $O(n^{(1/d)})$  in which  $n$  is the number of peers in the overlay network and  $d$  is the number of dimensions of the CAN overlay network.

Many efforts have been spent on building topological-aware structured overlay networks. A *topological-aware* overlay construction method uses proximity information in the underlying physical network to map peers which are close to each other in the physical network into close locations in the overlay network. Topologically-aware CAN [79] suggests using landmark nodes to group the CAN nodes into the right zone in the Cartesian coordinated space. eCAN [98] builds a hierarchical CAN on top of the topologically-aware CAN construction protocol. Xu et.al [97] tried to improve the routing efficiency of structured networks by building an auxiliary expressway network, which takes advantage of the AS (autonomous systems) in the Internet.

The research in peer-to-peer overlay construction is an important building block of peer-to-peer cycle sharing system. However, only to leverage the existing overlay construction protocols may not be adequate. The reason is that most of the existing overlay construction protocols are motivated by peer-to-peer file sharing systems, but file sharing systems and cycle sharing systems differ in many ways:

First, the resources shared are different and they should be organized according to their distinct characteristic. For example, the usage pattern of files does not greatly influence the information sharing; however, the usage pattern of the CPU cycles greatly influences the scheduling in the P2P cycle sharing systems.

Second, the two types of applications place different requirements with different priorities. The peer-to-peer file sharing system emphasizes routing efficiency since fast communication is needed for lookup and transport. However, in the cycle sharing system, routing efficiency is not as important for typical cycle sharing applications in which computation time dominates and the data communication is infrequent. An important requirement for cycle sharing systems is users' willingness and ability to share cycles. Therefore, new overlay network construction protocols need to be designed to organize peers according to their available cycles and to support efficient scheduling strategies.

## 2.4 Resource Discovery in Peer-to-Peer Networks

The classical resource discovery problem in P2P networks is the problem of discovering dynamically changing available resources in a scalable matter. In a small-scale, static environment, it is adequate to use a centralized directory server keeping track of all the available resources. In a peer-to-peer environment, not only is the central server approach non-scalable, often there is no clear incentive for a third-party to provide such a service. Therefore a peer-to-peer system needs to use a fully distributed, self organizing resource discovery method.

### 2.4.1 Resource Discovery in Unstructured Peer-to-peer Networks

In an unstructured peer-to-peer system, the simplest resource discovery method [23] uses an expanding ring search. The request is sent to direct neighbors, and if unsatisfied, the client increases the searching scope, which may cause a high message overhead. A simple improvement to expanding ring search is random walk [70]. With random walk, the node sends a query to a group of random neighbors instead of all neighbors, and on receiving the query, the neighbor will forward the query to a group of their random neighbors. The message overhead of random walk is lower than expanding ring search, but still high.

Many works have been proposed to increase the searching efficiency in unstructured overlay networks, but all focused on improving performance for locating files but not for general types resources including cycles. Several works suggest using superpeers which function as a rendezvous point or directory server [56, 85, 20] as we have mentioned in the overlay construction section. In order to lower the network traffic, queries from the leaf nodes are handled by the super nodes and super nodes cache indexes of file on the leaf nodes. Some work [43, 91] focused on improving the request forwarding methods, based on the assumption that clients can be grouped according to their interests. In addition, a few papers propose to use non-forwarding methods [99].

### 2.4.2 Resource Discovery in Structured Peer-to-peer Networks

Unlike search in unstructured network, a structured peer-to-peer system, such as [78, 93, 83, 102], directly maps content to locations in the overlay network using a consistent hash function and the content is retrieved using the same hash function. Instead of doing a flooding-based search in the unstructured network, the request is

directly routed to the location, where the content is stored. For example, the content stored in a CAN overlay network is hashed into the coordinates of a point (the key of the content) in the Cartesian space. The node that owns the area containing that point stores the content or real address of the content. When a node wants to locate some file, it sends out a query to the node, which owns the key. On receiving the query message, each node forwards the query to the neighbor, which has the shortest Cartesian distance to the key. This procedure repeats until the query reaches the node that owns the key. Figure 2.2 demonstrates the content location in CAN: the key to locate is  $(0.11, 0.9)$ , which is owned by node A according to the rule.

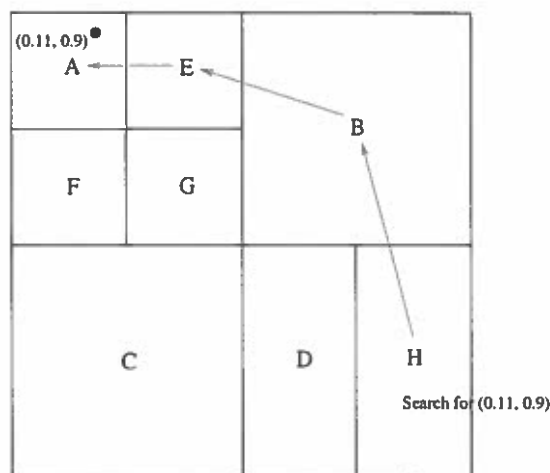


FIGURE 2.2: Resource discovery in CAN

Although there are a large number of resource discovery solutions proposed for file sharing peer-to-peer system, the question of how to effectively and efficiently locate idle machines in a fully distributed peer-based desktop grid system remains an open question. It is due to the inherent difference between file sharing systems and cycle sharing systems. A comprehensive study needs to be performed to understand whether the generic search methods such as expanding ring search and random walk are effective in peer-based desktop grid systems.

- The availability and amount of resources in cycle sharing system is much more dynamic than in a file sharing system. In the latter system, once a file appears, it remains in the system as long as the owner of that file stays. Also the file can be replicated in strategic locations and linger a long time, even after the original owner leaves the system. However, the availability of CPU cycles on host machines changes over time.
- The incentives and model of volunteer behavior will be different in the two systems. Users may be willing to allow others to download files, provided the bandwidth consumption is small. However, the same group of users may not be so willing to let others share CPU cycles when they are working on their own computer. They may require that foreign jobs leave their machine or sleep with the lowest priority when the owners of the machine reclaim it.
- In a cycle-sharing system, multiple resources, such as CPU, memory and network connection etc., need to be taken into consideration, while in file-sharing system there is only one type of resource concerned.

## 2.5 Cycle Sharing Systems

There are many existing cycle sharing systems, which can be divided into four categories: institution-based load sharing systems, Internet-based computing projects, web-based computing and existing peer-based desktop grid systems according to the scale of the systems and the infrastructure used by the systems. Figure 2.3 briefly illustrates the relationship between peer-base cycle sharing systems and their ancestor systems.



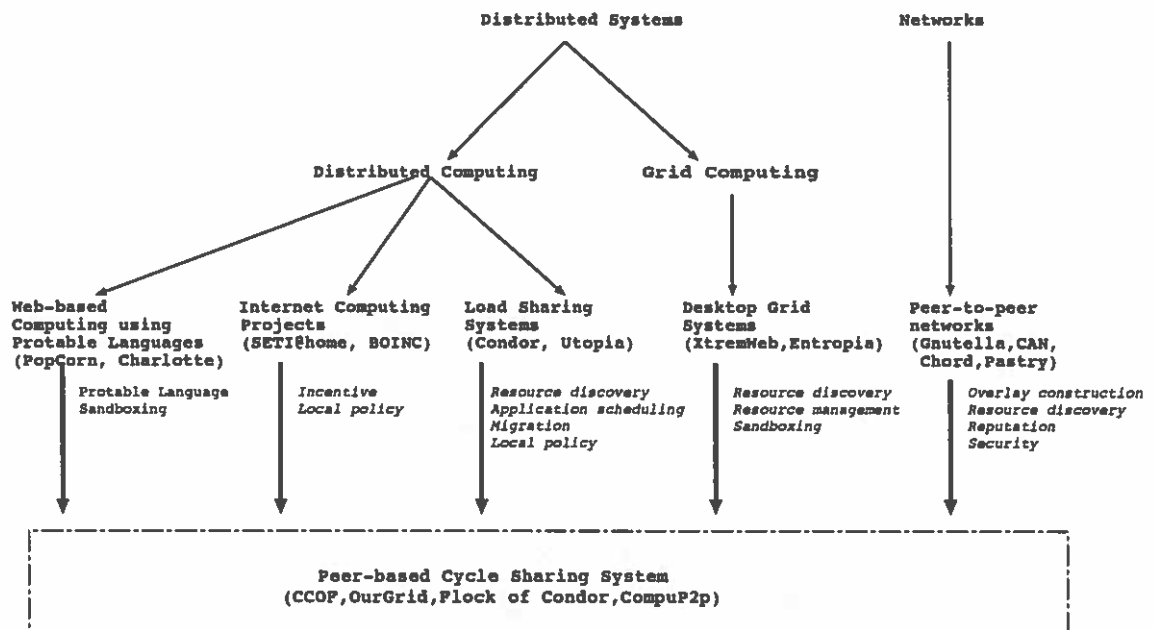


FIGURE 2.3: Relationship between peer-based cycle sharing systems and their ancestor systems

### 2.5.1 Institutional-based Load Sharing Systems

A key area of research that is related to peer-based desktop grid systems is that of institutional-based load sharing [87] in distributed computing systems. The goal of *load sharing* in distributed computing is to distribute the load across machines in the system and therefore reduce response time. Load sharing realizes this goal by moving jobs from heavily loaded nodes to lightly loaded nodes in the distributed system. This body of work developed during the late 1980s and early 1990s in the context of networks of workstations within a single institution, examples include Legion [74], Sprite and Condor [63]. As with peer-to-peer systems, the idea of harnessing idle cycles evolved naturally from the earlier idea of file sharing and data sharing (i.e. network file systems).

Condor [63] is the most famous classical load sharing systems. Condor was first installed as a production system in the University of Wisconsin Department of Computer Sciences nearly 15 years ago. Condor aggregates idle cycles within one institu-

tion and provides automatic scheduling of tasks. Condor uses a centralized match-making server to match available hosts with client requests. In this non-dedicated, opportunistic environment, whenever a host is judged as available and there is a client request for extra cycles, Condor can schedule a job on that host. When the host becomes busy again, Condor can preempt the foreign tasks and reinstate them later using checkpointing schemes.

To attract users to stay in the system, Condor allows clients to control their local resource using policies expressed in the profile of the hosts, including time slots when the host cannot accept foreign jobs, the CPU threshold below which the host can accept foreign jobs and whether the host can run foreign jobs when there is recent keyboard activity. Condor also allows users to manually create groups of people they trust and limit the access to those people only.

Over time, Condor has evolved with new techniques, changing from a load sharing system within one institution to a load sharing system across different administration domains [33, 94]. An augmentation to Condor architecture using peer-to-peer methods was recently proposed by Butt et.al [12], which will be discussed in the peer-based desktop grid systems.

From the above description, one key observation is that institution-based load sharing systems heavily rely on the correct configuration of the central servers, which take charge of resource management, resource usage negotiation and scheduling. Also, the institution-based load sharing systems has a strict membership management scheme: Only users within participating institutions can access the resource. An open peer-based desktop grid systems can circumvent these drawbacks with its fully distributed architecture and access to ordinary users.

## 2.5.2 Internet-based Computing Projects and Web-based Computing Projects

Internet-based computing projects, such as SETI@home [84] and BOINC [90] use a client-server model to distribute tasks to volunteers scattered in the Internet. The cycle donators download and install the software, which can do the computation under a screen saver model or background model. The host connects to the central server when it wishes to download new code and data. Because an Internet-based computing project requires manual coordination from a central server, it experiences downtime due to surges in client requests. SEIT@home and Folding@home have already demonstrated that was a serious performance bottleneck. Also, the goals of such systems are to support one particular application, so only one client can take advantage.

Web-based computing projects, such as PopCorn [15] and Charlotte [6], are motivated by the portable, secured Java language and the web techniques. They use client-server architectures, and all the programs are implemented in Java. When a client wants to recruit some workers to compute its applications, it advertised the URL of the work on a central website. The users who are willing to help will browse the active programs on the website and manually download the program to run on their machines. This type of infrastructure is un-scalable and very difficult for user to deploy and use.

There are many common features among Internet-based computing projects and web-based computing projects, and thus they share same flaws. The server in this type of system is performance bottleneck and single point of failure. In addition, both of the systems require a lot of human interference in configuring the server and manually downloading the program for the systems to function correctly.

### 2.5.3 Peer-based Desktop Grid Systems

Traditional Grid computing [38, 27] is an infrastructure that integrates supercomputers, networks, databases and scientific instruments from multiple sources to form a virtual supercomputer on which users can work collaboratively. With the flourishing of peer-to-peer networks, Grid systems began to evolve into peer-based desktop grid systems. A peer-based desktop grid systems harness idle cycles on desktop machines using a peer-to-peer overlay network.

Peer-based desktop grid systems belong to the category of desktop grid systems. The desktop grid systems include both master-slave desktop grid systems and peer-based desktop grid systems. The master-slave desktop grid systems use the same master-slave/client-server model as the Internet-based computing projects. One of the new desktop grid systems, the XtremWeb [17, 16, 30] project, has two types of implementation: a traditional grid computing infrastructure or a SETI@home style infrastructure. The latter requires a central sever as coordinator. Clients send tasks to the server and the tasks are batched in a task queue. When the worker (donor of cycles) is idle, it requests new tasks from the server. Entropia [32, 14] desktop grid system designed by the AppleS group [18] also uses central servers for resource discovery and management. The focus of the project is to build a virtual machine using sandboxing technique within an enterprise or an institution.

The peer-based desktop grid systems are motivated by the new peer-to-peer techniques. Foster et.al.'s work [39] and Ledlie et.al.'s work [61] analyzed the differences and similarities between the two fields and predicted that the two fields will eventually converge. Also in pragmatic aspect, some of the desktop Grid computing projects, such as OurGrid [3, 26], Personal Grid [46], Flock of Conder [12, 13, 11], and SHARP [42] are designed and implemented using peer-to-peer methods for resource location. We also propose a general peer-based desktop grid architecture CCOF [67], which is described in details in chapter 3. Table 2.3 summarizes existing peer-based desktop grid systems.

TABLE 2.3: Existing Peer-based Cycle sharing Systems

Project	Focus	Overlay Network	Resource Discovery	Application Scheduler
OurGrid [3]	Fairness, Scheduling for Bag-of-Tasks	Unstructured	Flooding	Hosts with most credits
SHARP [42]	Security, Fairness	Unstructured	Advertisement, Link-state style Gossip	Ticket-based bartering
Flock of Condor [12, 11]	Easy assembly, Fairness	Pastry Structured	Advertisement, Broadcasting	First available
Personal Grid	Assembling, Matchmaking	Gnutella style Unstructured	Rendezvous points	First available
CCOF [67]	Scheduling, Result Verification	CAN/Pastry Structured	Rendezvous points	Multiple criteria, Migration
CompuP2P [45]	Incentives, Economic model	Pastry Structured	Rendezvous points	Lowest financial cost

- OurGrid [3] is built on JXTA and uses the JXTA resolver as supernodes to map client requests to available resources and therefore heavily relies on the flooding-based performance of JXTA [25].
- SHARP [42] is aimed at secure resource sharing, which is implemented in Planet-Lab [76]. It uses a static unstructured overlay formed by nodes in the PlanetLab testbed.
- Personal Grid [46] proposed to use a two-hierarchy overlay network similar to gnutella network [56] to build the cycle sharing system. The system clusters nodes according to the local networks they belong to, and rendezvous points in each cluster are in charge of resource discovering and scheduling.
- Flock of Condor [12, 11] reported research in peer-to-peer cycle sharing at Purdue University. In Flock of Condor, the cycle sharing community is built on Pastry. The system is also a heavyweight system, which maintains a fine grained strict cycle-for-cycle fairness. The system requires modification to the compiler

strict cycle-for-cycle fairness. The system requires modification to the compiler as it inserts beacons into the program at compile time for checking job progress and keeping account of cycle usage.

There is also research focused on cycle bartering systems and auction strategies in this type of systems. CompuP2P [45] came up with a peer-to-peer cycle bartering system and studied auction strategies in this type of systems, which is built on Pastry overlay network [83].

## 2.6 Resource Discovery and Application Scheduling in Peer-base Desktop Grid Systems

There are many dimensions of designing the peer-based desktop grid systems including overlay construction, resource discovery, scheduling, incentive and trust. The most relevant work to my research is about resource discovery schemes and scheduling for faster turnaround.

### 2.6.1 Resource discovery in Peer-based Desktop Grid Systems

The goal of resource discovery is to identify available resources in the system. There are many resource discovery methods proposed for peer-to-peer file sharing system [22, 92, 60, 101, 44], since peer-to-peer file sharing system is the first peer-to-peer system ever created and remains the most popular applications. While many of them are tailored to content-sharing systems, the generic resource discover method such as expanding ring search, random walk, advertisement-based search may be leveraged into cycle sharing systems. With expanding ring search, a client sends out queries to all its neighbors for resource discovery. If the search fails, the client expands the search scope by one and the sends out query again. With random walk, a

client sends out queries to a random of neighbors and these neighbors will forward the queries to their random neighbors. With advertisement-based search, hosts advertise their available resource periodically, and clients cache such information and consult their local cache when resources are needed. Peer-based cycle sharing systems use these solutions without a systematic study on how well these solutions scale to a large-scale dynamic system.

The research conducted by Iamnitchi proposed peering multiple sites in a Grid computing infrastructure [53]. She conducted a study on how to satisfy resource discovery requests for a variety of types of resources. The research compared random walk with a learning-based strategy and a history-based strategy to make the request-forwarding decision, and analyzed the latency (in hops to satisfy resource requests.

The project Flock of Condors at Purdue [12] used an advertisement-based approach, in which a Condor server sends out resource information for its site in a limited scope. On receiving the information, Condor servers of other sites cache and then select the candidate Condor flock when needed. The system is static and the paper does not present direct measurements of how this type of resource discovery performs. Also it does not compare their method with other types of search methods.

The SHARP project [42] proposed an architecture for secured resource sharing. It uses a link state style protocol to propagate resource information, and then nodes compute the route to the available resource. This method is not scalable in a large, highly dynamic environment with the heavy traffic of resource information propagating in the system. The emulation is on a static and small scale overlay network (PlanetLab), so it cannot demonstrate how the protocol works in a large scale peer-to-peer system.

The Personal Grid project [46] uses a rendezvous point based resource discovery methods. The system clusters nodes according to the local networks they belong to, and rendezvous points (supernodes) in each cluster are in charge of resource discovering and scheduling. Ordinary nodes report their status to the rendezvous points and

rendezvous points function as matchmaking servers to match clients' requests with available hosts. The study does not provide a dynamic rendezvous points selection methods which is core to the success of this type of systems: It is unclear how to cluster those nodes which are not in any local networks, and how to balance load among rendezvous points.

There are recent work using DHT overlay network for resource discovery in Grid. NodeWiz [7] connected the directory servers of individual organizations using DHT methods. Y. Huang et.al [52] described DHT-based resource management service and its API to replace the current web-based Grid services. Both of them are not designed for a fully distributed peer-based cycle sharing system, and it is unclear whether they can work without the institutional supports. M. Hauswirth [48] suggests that users can store the description of their work in DHT overlay network using the number of CPU cycles as the key. Resource provider then look for jobs which they are willing to process. This rigid method requires precise prediction of job running time, which is often not feasible. Felix [51] uses DHT overlay network to store ontology information of the machine, which can only handle discovery for static resource information such as CPU types and operating systems.

Recent studies [21, 4] propose to use DHT overlay network to support point-in-range queries for grid resource discovery. Andrzejak et. al. [4] use space-filling curves to perform range query processing in P2P environments based on CAN overlay network. L. Chen et. al. [21] studied several different methods to match virtual information space to physical nodes with replication schemes to reduce search time and to achieve load balance.

One observation is that the proposed resource discovery methods based on DHT overlay network are effective in discovering static resource information such as search for any Linux machine with CPU speed larger than 500MHZ. However, a missing piece is search for dynamic resources such as available cycles, which is important in peer-based desktop grid systems centered with the goal of cycle-sharing. There are



two aspects in answering this question. First, how do the generic research methods perform in discovery cycles, these methods can handle dynamic information well as changes of resource information only impacts the local hosts, in contrast to DHT overlay network, where keys need to be updated on all replicas. Second, whether the DHT overlay network can be leveraged to store coarse-grained cycle profiles, which trades precision for a better search speed and search overhead.

### 2.6.2 Application Scheduling in Peer-based Desktop Grid Systems

The function of the application scheduler is to choose the best host to schedule the job on and ship code to that host. All of the previous Internet-scale cycle sharing systems use the simple and straightforward scheduling method, that is to choose the first available host and the job will stay on that host during its entire lifetime.

To my best knowledge, none of the previous work has addressed the fast turnaround scheduling problem in a scalable peer-based cycle sharing system. The most closest work so far described methods for improving fast turnaround in institutional grids. The most popular approach is to improve turnaround time via replication. With replication, more than one duplicated copies of the task are submitted to the system, to prevent one node from slowing the whole computation. Y. Li et. al [62] analyzed performance improvement via replication using a centralized scheduling server. D. Kondo et. al [57] evaluated the performance of replication with rescheduling when hosts withdraw cycles. Replication scheme is not desirable when the system already has a heavy load, as the duplicated copies will further burden the system. A recent paper [59] described scheduling for rapid application turnaround on enterprise desktop grids. The computation infrastructure used a client-server model, in which a server stores the input data and schedules tasks to a host. When the host becomes available, it sends a request to the server. The server keeps a queue of available

hosts and chooses the best hosts based on resource criteria such as clock rate and number of cycles delivered in the past. The work did not considering migration schemes. Moreover, this work is limited to scheduling within one institution and it uses a client-server infrastructure, while scheduling in a large scale fully distributed peer-based cycle sharing system is much more complicated and challenging.

The research in peer-based desktop grid system touches on research in both peer-to-peer networks and distributed computing. It uses the peer-to-peer overlay network and distributed resource discovery techniques from the peer-to-peer networks, and it uses the concept of cycle sharing, opportunistic scheduling and local admission policy from distributed computing systems. Taking advantage of both worlds, the unique peer-based desktop grid systems chart many new research questions. The previous peer-based desktop grid systems have focused on fairness, best-effort scheduling, and secured resource sharing. The area of fast scalable resource discovery and scheduling for fast turnaround remains an unexplored area which directly related to users' satisfaction with the peer-based desktop grid systems. My dissertation represents pioneering work to help provide solutions to these open questions and pushes the envelope of research on cycle-sharing in the Internet.

## CHAPTER 3

# A General Architecture for Peer-based Desktop Grid Systems

We propose a general architecture for peer-based desktop grid systems which uses the classical peer-to-peer paradigm. The goal is to build a flexible, distributed, modularized system, which does not require central services. In contrast to Grid computing [37, 94] and other institution-based cycle-sharing systems [63, 69, 106], our architecture targets a more open environment, which is accessible to ordinary users and does not require strict institutional based membership management. However, a federated institutional-based system can use a similar architecture, peering different sites using a peer-to-peer overlay network, in which each institution joins as a peer.

### 3.1 CCOF: a General Architecture for Peer-based Desktop Grid Systems

All of my dissertation research is based on CCOF (cluster computing on the fly) [67], a peer-based cycle sharing framework, which I helped to develop at the University of Oregon. CCOF is designed to support a large scale, open, and dynamic

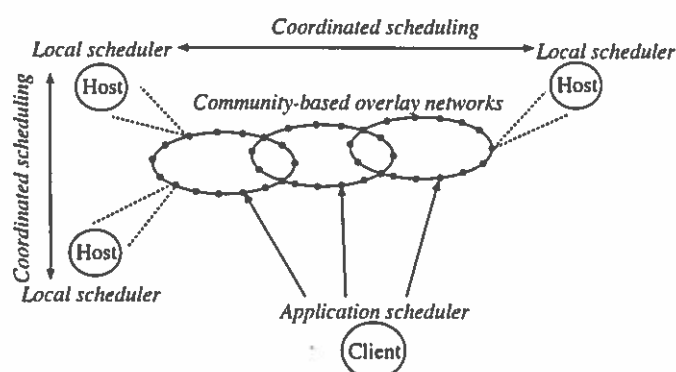


FIGURE 3.1: CCOF Architecture

cycle sharing community spread throughout the Internet. The key components of the CCOF framework include overlay construction, resource discovery, application scheduling, local scheduling, and coordinated scheduling. Other components such as reputation system, accounting, and performance monitoring can be added to the CCOF framework to further improve the QoS of the system. Figure 3.1 shows the architecture and the cycle sharing community of CCOF.

In CCOF, peers join a cycle sharing community overlay network to request or donate cycles based on their interests. Candidate hosts are discovered through resource discovery schemes initiated by the clients when they need idle cycles for their application. The application scheduler on the client chooses the best among the candidates to schedule the jobs on and ships the codes to the selected host(s). Figure 3.2 shows how a client discovers and then sends a request to a host when it submits a job to CCOF. Depending on the local scheduling strategies, a host may preempt the foreign tasks or suspend the foreign tasks when it becomes busy. Figure 3.3 shows how a host handles a client's request using its local scheduler. When results are returned from hosts, clients may utilize CCOF's result verification techniques to check the correctness of the results returned by the potentially untrusted peers.

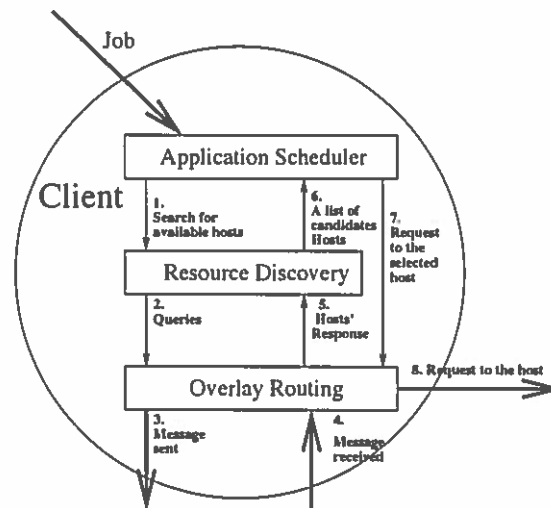


FIGURE 3.2: Client submits job to CCOF.

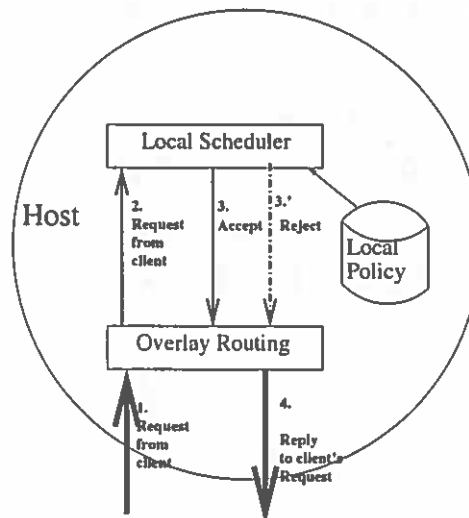


FIGURE 3.3: Host receives request from client.

- **Overlay Management** This component organizes communities of hosts willing to donate cycles. Communities may span multiple organizations, such as a collaborative research project among several research groups. Chess enthusiasts, or participants in the SETI@home project [84] may form a community based on their hobbies or a spirit of volunteerism.

One way to organize such communities is through the creation of overlay networks. The overlay management handles hosts join and leave, and detects and repairs the overlay network when there is host failure. It also provides routing schemes for peers to communicate with each other. CCOF overlay management leverages the generic peer-to-peer overlay network construction protocols such as CAN [78], Chord [93] and Pastry [83] by also considering information such as trust, performance and geography.

- **Resource Discovery** The resource discovery component is used by clients to collect resource information of the hosts and discover idle hosts. The resource information about a host collected by a resource discovery scheme may include the CPU speed, CPU utilization, memory and the type of operating system.
- **Application Scheduler** The application scheduler is responsible for selection of hosts for a P2P computation from a pool of candidates, to export the code to the hosts, and to manage and verify results. Application scheduling requires an analysis of both the application's needs and the nature of the offered resources to achieve the best performance. The application scheduler may also use techniques to improve the Quality of service such the migration schemes for fast turnaround and checkpointing for fault-tolerance.
- **Local Scheduler** The local scheduler tracks idle cycles and negotiates with the application scheduler using local criteria for trust, fairness, and performance to decide which tasks to accept. It also interacts with its own native scheduler to inject those jobs into the scheduling queue.

- **Coordinated Scheduling** A meta-level coordinated scheduling may also be implemented to coordinate among local scheduler or application scheduler to enforce security policy or long-run fairness through sharing of trust ratings.

## 3.2 Suitable Applications for Peer-based Desktop Grid Systems

We identified three types of suitable applications in Peer-based desktop grid systems: workpile applications, tree-based search applications, and point-of-presence applications, which are briefly introduced in this section. Their scheduling needs are starkly distinct, calling for individualized scheduling services that are tailored to those needs. Among them, the most popular applications supported by the current Internet-wide cycle sharing systems such as SETI@home and BOINC is the workpile applications. Therefore, my research is focused on scheduling for workpile applications, and I will describe the character of workpile tasks in next section.

**Workpile Applications.** These applications consume huge amounts of compute time under a master-slave model. Each host computes intensively, and then returns the results back to the master node. The workpile application is “embarrassingly parallel” in that there is no communication at all between slave nodes. The workpile applications can be further divided into infinite workpile tasks, which runs for a long period time and deadline driven workpile tasks, which has a moderate need of cycles and requires the result be returned fast. While many scheduling strategies are suitable for infinite workpile jobs with and without deadlines, the urgency of deadlines calls for more efficient approaches for discovery and scheduling of cycles. Details of workpile tasks are described in next section.

**Tree-based Search Applications.** This class of applications requires substantial compute cycles, with loose coordination among subtasks requiring low communication overhead. Distributed branch-and-bound algorithms, alpha-beta search, and recursive

backtracking algorithms are used for a wide range of optimization problems; these computationally intensive state-space search algorithms are ideal candidates for P2P scheduling.

Distributed branch-and-bound algorithms use a tree of slave processes rooted in a single master node. The tree dynamically grows as slave processes expand the search space and is dynamically pruned as subspaces leading to costly solutions are abandoned. There is a small amount of communication among slave nodes to inform other slaves of newly discovered lower bounds.

The scheduler manages the dynamic population of host nodes by continuously providing new hosts while the search tree is growing. It must also support communication among slave nodes, either indirectly through the master or directly from slave to slave.

**Point-of-presence applications.** PoP applications typically consume minimal cycles but require placement throughout the Internet (or throughout some subset of the Internet).

The dispersment of tasks from a PoP application is driven by specific requirements of the job. For example, distributed monitoring applications (security monitoring, traffic analysis, etc.) require widely and evenly distributed placement as well as placement at strategic locations. Testing of distributed protocols requires placement of test-bots dispersed through the Internet in a manner that captures a variety of realistic conditions with respect to latency, bandwidth, and server performance. In addition, security concerns must be addressed by limiting communication only to the set of PoP tasks.



### 3.3 Workpile Applications.

The type of applications I focus on are known as *Workpile (Bag-of-tasks)* jobs, those requiring large amounts of CPU cycles but little if any data communication. A workpile job may be a single, long-running task, or a (large) collection of long-running tasks. These CPU intensive tasks are referred to as as *embarrassingly parallel* tasks in the parallel processing community.

Workpile applications are ideal candidates for a peer-based cycle sharing system, because the scheduling overhead of such applications, which includes the resource discovery overhead and the cost of transferring the code and data over the Internet, is negligible compared to the total runtime of the applications. The code size of many long running applications is small and these applications require minimal data communication. For example, the average data moved per CPU hour by users of SETI@home is only 21.25 KB, which is feasible even for users with slow dial-up connections. With respect to program size, Stanford Folding is only about 371KB, and SETI@home is around 791KB (because it includes the graphical interface for the screensaver). These applications run for a long time. Using the same SETI@home example, the average computation time of each job is over 6 hours (on an Intel x86).

Examples of workpile applications include state-space search algorithms, ray-tracing programs, gene sequencing and long-running simulations. Table 3.1 summarizes prominent workpile applications.

Most cycle sharing systems are best effort and presume that the client will be satisfied to complete the results within a reasonable amount of time. However, many users' applications may have higher performance requirements for faster turnaround time and higher throughput. Some of the above applications may have real time constraints, such as weather forecasting, or deadlines such as scientific simulations that must be completed in time for a conference submission. In general, productivity

**TABLE 3.1:** Examples of workpile applications

Application	Description	Running Time
SETI@home	Search for Extraterrestrial Intelligence by analyzing radio telescope data	Infinite
Intel Philanthropic	life science research programs dedicated to specific diseases, from prostate cancer to Parkinson's disease	Years
Distributed.net	Brute force key decryption	Years
Folding@home	Study protein folding, misfolding, aggregation, and related diseases	Days
State space search graphics	simulated annealing, branch-bound genetic algorithms	Hours to days
Graph algorithms	Geometrical transformations, partition, shortest path, minimal cut	Hours to days
Simulations	Numerous	Hours to days
Image processing graphics	ray tracing, sky curvature, 3-D mosaicing	Hours

for the users and efficient use of the cycle sharing system is enhanced by higher performance scheduling. These needs motivate my research into resource discovery and scheduling for faster turnaround in peer-based cycle sharing systems.

## CHAPTER 4

# Resource Discovery in Peer-based Desktop Grid Systems

### 4.1 Introduction

The resource discovery methods in peer-based desktop grid systems locate available idle hosts for clients in needs of extra cycles. The chapter of the dissertation is a systematic analysis of resource discovery methods for discovery of idle hosts to serve as candidates for cycle sharing. Through a careful simulation analysis of classical rechniques used in peer-to-peer networking, we found that rendezvous point provides the fast resource discovery needed for fast turnaround scheduling.

Resource disocvery is a unique challenge for an open peer-based desktop grid system such as the CCOF infrasture described in chapter 3, because the set of participating hosts is potentially very large and dynamic. With its open nature, peer-based desktop grid systems will be of much larger scale and be much more dynamic than the traditional institutional-based distributed systems and traditional GRID systems. In traditional distributed systems, central servers function as matchmakers to match the request with the available resources or representatives for resource negotiation [63, 65, 9]. This central server approach is not scalable and cannot be

adapted to peer-based desktop grid systems environment, as the central servers are performance bottlenecks and traffic hot spots. Also there is no clear incentive for a third party to provide such institutional support in this open peer-to-peer system.

The amount of available resources in peer-based desktop grid systems is highly dynamic. Like any other peer-to-peer system, peers may join and leave the system at any time. Furthermore, the amount of available resources in peer-based desktop grid systems may change much more quickly and dynamically over time than in traditional content sharing peer-to-peer systems. In the latter systems, once a file appears, it remains in the system as long as the owner of that file stays. Also the file can be replicated in strategic locations and linger a long time, even after the original owner leaves the system. However, the crucial resource of CPU cycles in peer-based desktop grid systems changes over time and cannot be replicated. Users may be willing to allow others to download files, provided the bandwidth consumption is small. However, the same group of users will not be so willing to let others share CPU cycles when they are working on their own computer. The foreign jobs should leave that machine or sleep with the lowest priority when the owner of the machine reclaims it. The resource discovery strategy used by peer-based desktop grid systems has to be adaptive to this ever-changing environment.

The rest of this chapter is organized as follows. Section 4.1 describes the research methodology and assumptions. Section 4.2.5 presents the preliminary simulation results and Section 4.3.2 concludes and discusses future work.

## 4.2 CCOF Resource Discovery Model

In this section, we describe the CCOF resource discovery model, which includes the profile based model that describes each host's idle cycles; the dynamic behavior of hosts; and the search algorithms we study.

### 4.2.1 Dynamic Hosts

In CCOF, peers can leave and join at any time. When one peer joins the system, new idle cycles will be available to the whole system; when one peer leaves the system, the foreign task running on that peer will be stopped and that peer will inform the initiator of the task to reschedule or migrate the task.

In CCOF, peers can withdraw cycles when the user reclaims his machine, based on local policy. The peer cannot then be used to do computation, but it can still be used to relay messages for other peers.

In traditional institutional based load sharing schemes, migration is acceptable, because within one institution, the network latency between machines is low and the central server provides quick discovery of newly available resource; thus the migration cost is comparatively small. However, within CCOF, the peers are scattered throughout the Internet. Thus, there is no guarantee about the speed of locating new resources among the peers and transmission speeds may vary greatly. CCOF needs to be carefully designed to avoid unnecessary migration, due to lack of knowledge about the 'remote' host.

In an unreliable environment, peer machines may crash, and then the task dies. The application scheduler will detect the peer failure when it stops receiving heart-beat messages from that machine. The application scheduler then tries to restart the tasks on some other available host.

The results reported in this dissertation report the impact of hosts' withdrawing cycles from the system due to changes in users' daily schedule. This is a notable phenomenon in systems for sharing idle cycles, while it is not so visible in other types of peer-to-peer systems. Frequent changes in machine usage result in frequent resource discovery activities. Methods which are not scalable will then produce excess amounts of resource discovery message traffic.

## 4.2.2 Profile Based Model

We use a profile-based model to generate resource information about idle cycles on each host. The profile we use is based on the observation that people have daily routines for using their machines and that most machines are idle at night. For example, users may process email and browse the Internet in the morning; then their machines will be idle while they are in meetings or attending classes. Such usage patterns can be acquired manually by user input or automatically from a monitoring program running on the local host.

Currently, we only use profiles of available CPU cycles, however other resource information such as memory size and operating system information can be easily added into the profile as a separate attribute.

## 4.2.3 Workpile Applications

Workpile applications are those that consume huge amounts of compute time under a master-slave model in which the master gives out code to many hosts, each host computes intensively and then returns the results back to the master node. The workpile application is 'embarrassingly parallel' in that there is no communication among slave nodes.

## 4.2.4 Search Algorithms

We evaluate four different generic search methods. The probing based search methods such as expanding ring search and random walk search are widely used in peer-to-peer file sharing networks. However, they are often associated with a high message overhead when they are used to search for rare items. The four search methods can be used in both structured overlay network and unstructured overlay network. In our simulation study, we studied their behavior in an unstructured overlay network without losing any generality.

The search latency of those methods are bounded by the scope of the search or distance from non-rendezvous point to rendezvous point. However, if it fails to find an available host, the client has to start another search. It may take a long time to locate an available host, if the client has to search several times before it succeeds.

- **Centralized Search.** For comparison purposes, we also implement a centralized search algorithm, which we know is non-scalable. When a peer joins the system and it is willing to share the idle cycles, it reports its profile information to the central server. Clients send requests for cycles to the server. The central server then matches their needs with available idle hosts. With our simulation, the server always tries to find idle hosts near to the source of the task.
- **Expanding Ring Search.** When a client peer needs cycles, it sends the request for cycles to its direct neighbors. On receiving the request, the neighbor compares its profile to the request. If it is currently busy or the block of idle time is less than the requested block of time, the neighbor turns down the request; otherwise the neighbor accepts the request by sending the client peer an ACK message. If the client determines there are not enough candidate hosts to satisfy the request, it then sends the request to peers one hop farther. This procedure repeats until the client peer finds enough candidates to start the computation or the search reaches the search scope limitation.
- **Random walk search.** When a client peer needs cycles, it sends the request to  $k$  random neighbors. On receiving the request, like the expanding ring search, the neighbor then tries to match the request with its current status. If it has enough time to complete the task, it then sends back an ACK message. The neighbor also forwards the request to its  $k$  random neighbors, until the request reaches the forwarding scope limitation.

- **Advertisement based search (Ads-based search).** When a peer joins the system, it sends out its profile information to neighbors in a limited scope. The neighbors then cache such profile information along with the node ID. When a client peer needs cycles, it consults the profiles cached locally, and selects a list of available candidates. Since other clients may try to use those hosts at the same time, the client then needs to directly contact each of the candidates to confirm their availability. If a host is not available at this time, the client then tries the next host in the list, until the list is exhausted or the request is satisfied. There are several possible selection schemes, such as choosing the nearest available hosts, choosing the hosts with longest available time or choosing hosts with shortest matching available time block. Our simulation results show different selection schemes yield similar performance. The result we present in the simulation section uses the scheme of choosing the nearest available host.
- **Rendezvous Point Search.** Our rendezvous point search method uses a group of dynamically selected Rendezvous Points in the system for efficient query and information gathering. Hosts advertise their profiles to the nearest Rendezvous Point(s), and clients contact the nearest Rendezvous Point(s) to locate available hosts.

We assume an out-of-band Rendezvous Point selection/placement method, which guarantees that Rendezvous Points are geographically scattered in the peer-to-peer system. We note that dynamic placement of Rendezvous Points such that the system is balanced and a sufficient number of Rendezvous Points is within short distance to every peer is still an open problem.

When a peer is selected as a Rendezvous Point, it floods information about its new role within a limited scope. On receiving such a message, the peer adds the new Rendezvous Point into a local list and deposits its profile information on this new Rendezvous Point. When a peer joins the system, it acquires the



list of Rendezvous Points from the nodes it contacts or acquires the information through some out-of-band scheme. When a client peer needs extra cycles, if it has not already cached information about nearby Rendezvous Points, it queries its neighbors until it accumulates a list of known Rendezvous Points. The peer then contacts Rendezvous Points on the list one by one and each Rendezvous Point then tries to match the request with candidate hosts. This procedure repeats until the request is satisfied or all the known Rendezvous Points have been queried.

#### 4.2.5 Scheduling Strategies

After the application scheduler has discovered candidate hosts for the client job, it can choose hosts based on multiple criteria, such as trust value, performance ranking etc. In this simulation, we simplify the application scheduler and use only the availability of cycles as the criteria.

When a client fails to find enough resources to start the computation, the application scheduler can choose to give up or to try to reschedule the task at night, since hosts have maximal available time at night. This is similar to the notion of prime time v. non-prime time scheduling enforced by parallel job schedulers[66]. We also did simulations using exponential back-off scheduling. With that method, when the application scheduler fails to find enough resources, it retries after a random amount of time. If it fails, it can retry multiple times, each time with a doubled back-off time. The simulation results we present in this paper report the first two scheduling methods: no retry and retry at night (The performance of exponential back-off is comparable to retry at night).

After the client finds the idle hosts, in order to avoid unnecessary competition and migration, it then reserves blocks of time on those hosts to start the computation. If a host withdraws cycles, the host migrates the task.

## 4.3 Simulation

### 4.3.1 Simulation Configuration

Our CCOF resource discovery experiments are conducted using ns simulator. A power-law topology of 4000 nodes is used as the overlay topology, as current studies have shown that peer-to-peer systems exhibit power-law properties [82, 5].

During the simulation, peers in the system are divided into two non-overlapping groups. One group is the hosts, providing the idle cycles. The other group is the clients, needing extra cycles. The ratio of clients to hosts varies from 0.1 to 1.3. At light workloads, the hosts outnumber the clients; at heavy workloads, the clients outnumber the hosts.

We model the dynamic peer-to-peer environment by varying the probability that a given host will withdraw cycles from the system at any particular time of the day. The idle cycles on that host become unavailable to the system after that time; however the host still relays messages for the other peers.

We conduct a one-day simulation of CCOF. For each host, a 24-hour profile of idle time blocks is generated. The synthetic profile is generated in the following way: The machine is idle during the whole night (from 7pm to 7am); then for each time unit (one hour) during daytime, it is randomly decided whether the machine is idle or not. The probability of a machine being idle in one time unit is 0.3 in this simulation.

Clients submit random numbers of jobs into the system at random times during the day based on two different client request arrival distributions. Two probability distributions are used to model client request arrival patterns. With a uniform distribution, a client is equally likely to submit a job at any time of the day; with normal distribution, there is a peak load at noon.

The jobs are characterized by the number of processors needed and the length of time needed on each of the processors. The minimum requested time block is one hour, while maximum during the day is 4 hours and the maximum during the night is

6 hours. The number of processors ranges from one to a maximum of 10 % of all the peers in the system (400 in this simulation). The run time and number of processors are independently generated using exponentially decreasing functions(e.g number of jobs vs. number of processors or number of jobs vs. runtime).

We varied the search parameters for all of the search algorithms until we found those values that yield reasonable job completion rates and when possible, acceptable message passing overhead. For expanding ring search, the search scope is 5 hops in the peer-to-peer overlay. For advertisement-based search, the scope of advertisement propagation is also 5 hops in the overlay. Peers may contact up to 5 random neighbors and the search request can be relayed up to 12 hops with random walk search. Using the Rendezvous Point approach, 1 % of the nodes in the system are rendezvous points and a Rendezvous Point informs peers within 7 hops.

### 4.3.2 Simulation Results

Our evaluation of search algorithms is based on how successful they are in finding idle cycles and how scalable they are, based on the number of messages sent. In this simulation study, we are not concerned about latency of one search attempt, since it will be bounded by search scope. The metrics we use are the following:

- **Job Completion Rate.** Ratio of successfully finished jobs over total number of submitted jobs. The job completion rate can be broken up into the following:

$$\text{Job completion rate} = \text{first submission success rate} + \text{second submission success rate} - \text{migration failure rate}$$

where, first submission success rate is defined as the number of jobs successfully scheduled the first time they are submitted divided by total number of jobs; second submission success rate is the number of jobs successfully rescheduled at night divided by total number of jobs; and the migration failure rate is defined as the number of jobs failed after migration divided by the total number of jobs.

- **Message Overhead.** This is defined as the number of links traversed by all messages in the resource discovery procedure divided by total number of peers in the system. In this simulation, the message overhead is the total per peer over 24 hours. For Rendezvous Point, the message overhead also includes the messages for advertising the rendezvous points. Since the message count indicates the number of overlay links traversed, the count will be much larger in the physical network.
- **Average Distance.** This is the average hop count in the overlay from a client to the hosts on which its job is scheduled.

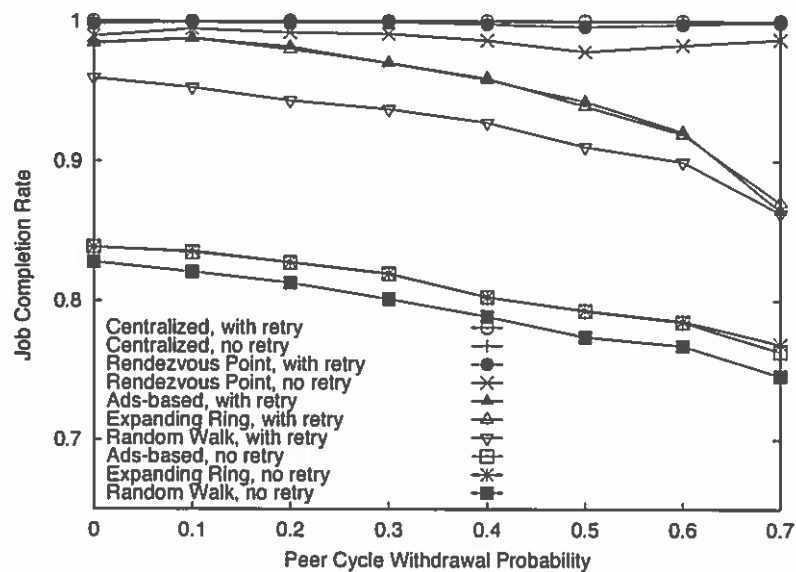
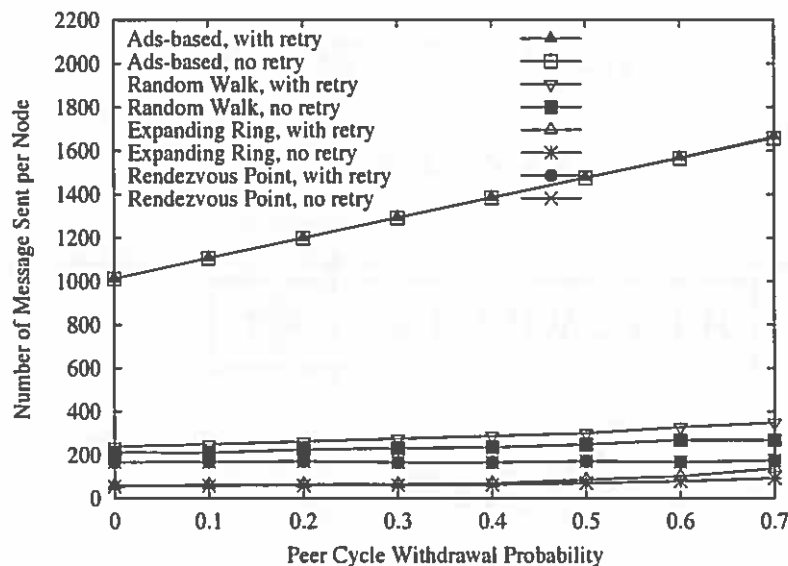


FIGURE 4.1: Job completion rate under uniform workload, when the ratio of clients to donors is 0.1.



**FIGURE 4.2:** Message overhead (for 24-hour period) under uniform workload, when the ratio of clients to donors is 0.1.

### Light Workload

This section shows the results under a light workload, with the ratio of clients to hosts set at 0.1 and assuming uniform client request arrival distribution. The results are similar with a normal arrival distribution under conditions of light workload.

Figure 4.1 shows that when there are abundant idle cycles, the job completion rate for Rendezvous Point without retry is very close or equal to the performance of a central server, with almost 100% completion rate over the full range of peer cycle withdrawal probability. Due to these two algorithms' inherent advantage in gathering knowledge of idle cycles, the client is guaranteed to find all or many of the hosts.

While the performance of expanding ring search and ads-based search are not as strong as the the first two methods, the job completion rates remain greater than 97% (with retry) for peer cycle withdrawal probability up to 30 %. (We note that since scope limit is 5 hops for both, their performance is basically identical). Random walk is the weakest of the algorithms with job completion rates around 95%. The simulation results also show that all the algorithms have less than 5% migration

failure rate. In general, all five search algorithms are not greatly impacted by peer withdrawal rates.

With a higher success rate on first submission, there is lower latency for scheduling the jobs. Using Rendezvous Point most clients (close to 100%) find enough hosts to start their tasks the first time they submit the tasks into the system. The other three methods only satisfy 75% to 83% of the tasks at first submission.

Figure 4.2 shows that the message overhead for Rendezvous Point is much lower than other techniques. The amount of messages sent for advertisement-based search is consistently high, since this technique uses flooding in a limited scope to advertise the profile information. The message overhead for random walk and expanding ring are comparatively small, as they are launched on-demand and under this light workload, the requests for extra cycles are infrequent.

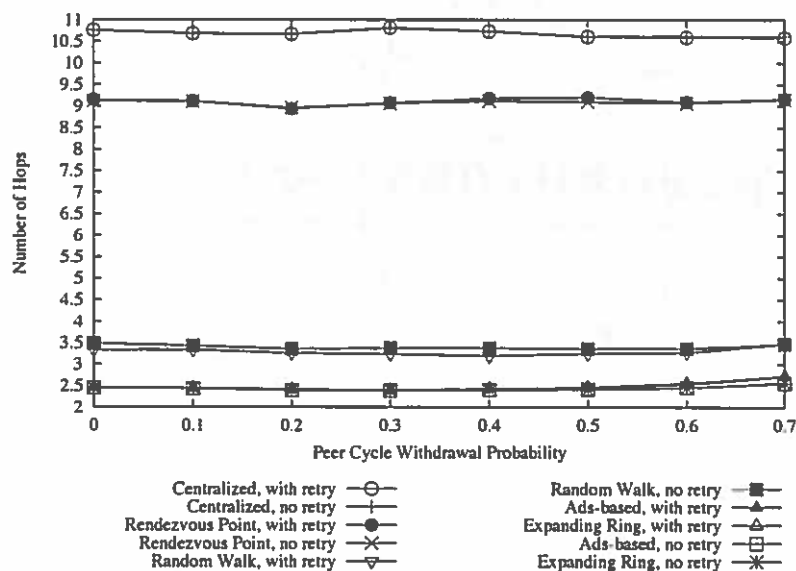


FIGURE 4.3: Average distance from clients to donors under uniform workload, when the ratio of clients to donors is 0.1

Figure 4.3 shows that random walk, expanding ring and ads-based search can only locate hosts in a local area, while the others can find hosts in a much larger range.

## Heavy Workload

This section shows the results under a heavy workload, with the ratio of clients to hosts set at 0.7. We first present results under uniform client request arrival distribution.

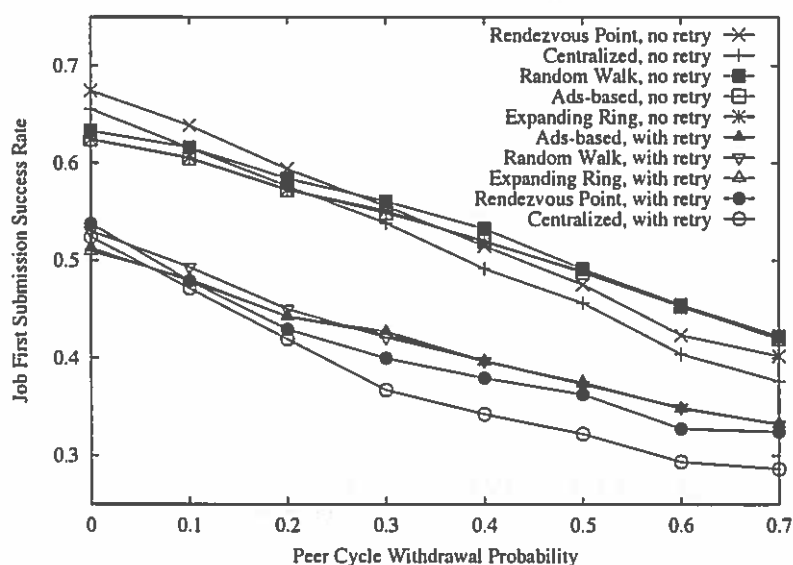


FIGURE 4.4: Job first submission success rate under uniform workload, when the ratio of clients to donors is 0.7

Figure 4.5 shows that the performance of all of the search methods degrade greatly under a heavy workload. We expected Rendezvous Point to consistently outperform the other search techniques. However, its performance drops below the others when the cycle withdrawal probability becomes higher than 0.1. We investigated this anomaly and found it is due to the ability of Rendezvous Point to discover more hosts in the system and thus be able to schedule larger task. In the CCOF environment, which satisfies requests on-demand, large jobs may block the smaller jobs from being scheduled. Table 4.1 compares of the average and maximum size of jobs scheduled by each algorithm with retry option turned on. Evidently, the average and maximum size of jobs scheduled by rendezvous point are much larger than the other three. We will discuss the nature and solution of this problem in section 4.3.2.

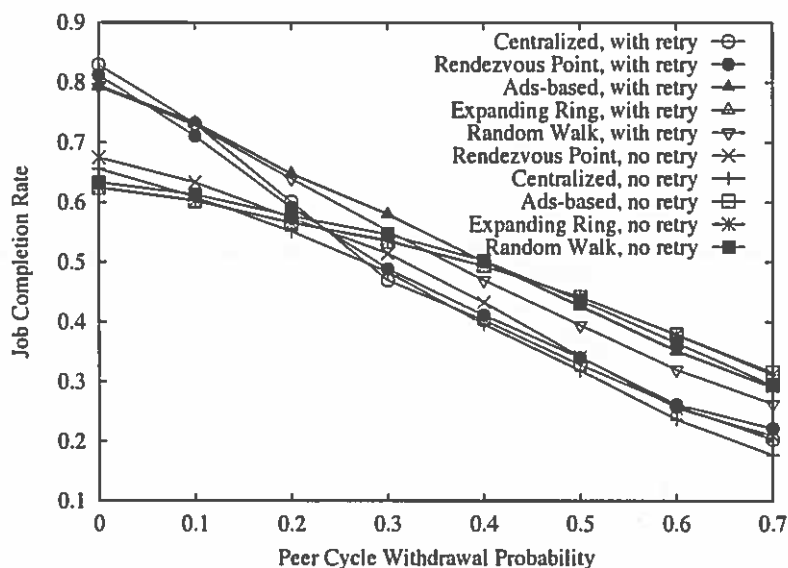


FIGURE 4.5: Job completion rate under uniform workload, when the ratio of clients to donors is 0.7.

TABLE 4.1: Average/Max size of jobs scheduled under uniform workload, when the ratio of clients to donors is 0.7. (Job size is measured in unit of processor-hours)

Peer cycle withdrawn probability	Central Server Avg-Max	Rendezvous Point Avg-Max	Random Walk Avg-Max	Expanding Ring Avg-Max	Ads-based Avg-Max
0	14.9-176	14.8-145	13.4-122	13.1-114	13.0-111
0.2	14.5-132	14.8-145	12.9-109	12.2-108	12.2-104
0.5	13.6-111	13.0-109	11.8-95	10.9-88	10.9-95

In Figure 4.4 the search algorithms are divided into two groups according to their success rate on first submission. The upper group does not allow retry and the lower group allows rescheduling at night. This graph shows a low first submission success rate for the second group. The jobs that fail to be scheduled during the day are then rescheduled at night, hurting the performance of jobs submitted at night.

The central server and Rendezvous Point algorithms have a higher migration failure rate than the other search methods (see Figure 4.6). Because the larger jobs occupy more processors, the likelihood that one of its processors will withdraw is higher than a small job.



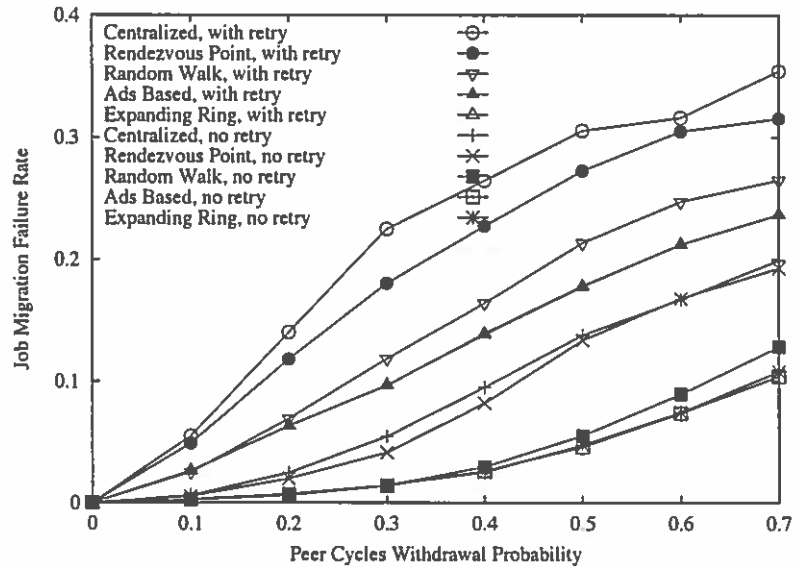


FIGURE 4.6: Job migration failure rate under uniform workload, when the ratio of clients to donors is 0.7

Rendezvous Point has the lowest and most stable message overhead (see Figure 4.8). However, expanding ring exhibits much higher message overhead for a heavy workload, compared to light workload (see Figure 4.2). The number of messages sent by it grows rapidly, because as cycle withdrawal probability increases, it needs to frequently probe in the peer-to-peer system with greater scope.

We conducted the same set of simulations with normal client request arrival distribution (See Figure 4.7). Algorithms that allow retry are significantly better than the corresponding algorithms without retry, while the difference is not as dramatic under the uniform distribution. After the peak arrival rate, there are fewer tasks to compete with the rescheduled tasks.

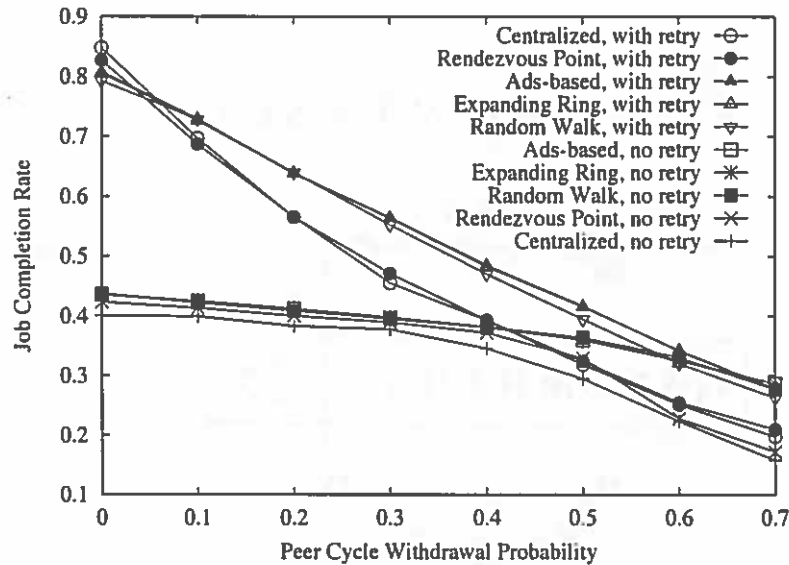
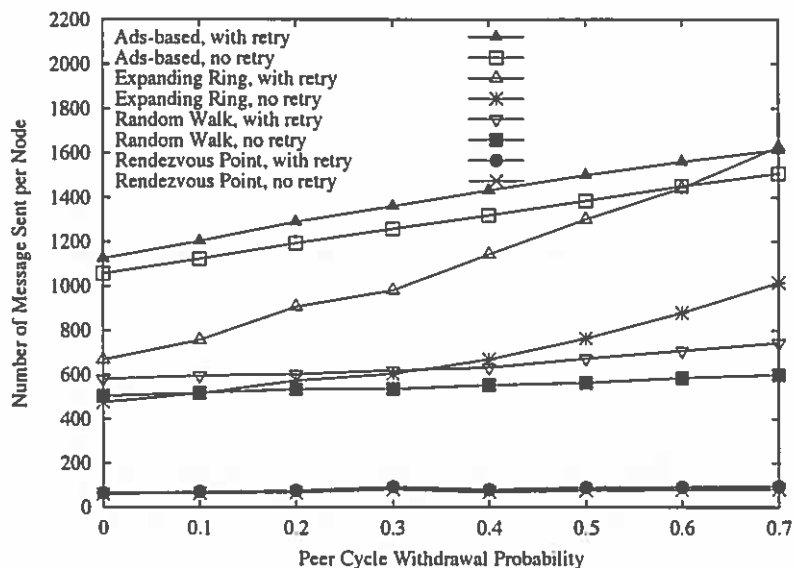


FIGURE 4.7: Job completion rate under normal workload, when the ratio of clients to donors is 0.7.

## 4.4 Conclusions

We make the following overall observations based on these simulations:

- The Rendezvous Point algorithm performs best overall with respect to all metrics and under both light and heavy workloads. The anomalous behavior that we observe under heavy workloads can be addressed through scheduling policies that avoid favoring large jobs. However preventing large jobs from starving small jobs is a hard open problem in a peer-to-peer environment. (This problem has been successfully addressed for parallel machines with a central scheduler using techniques such as backfilling.) In the chapter of conclusion and future work, we propose to address this problem in conjunction with mechanisms of fairness, QoS and security.



**FIGURE 4.8:** Message overhead under uniform workload, when the ratio of clients to donors is 0.7

- Under light workloads, while Rendezvous Point performs best, the other algorithms also perform well with job completion rates greater than 95%, when retry is allowed. They are stable over dynamic peer cycle withdrawal changes. Only ads-based incurs higher message passing overhead.
- Under heavy workloads, the job completion rate for all algorithms declines to values of 63% to 84% and drops quickly with increasing peer cycle withdrawal rates. The message overhead is still stable for Rendezvous Point, while it increases significantly for the other algorithms.

The simulation results show that rendezvous point algorithm performs best when considering both efficiency in locating available hosts and message overhead. The exceptional performance of rendezvous point algorithm is due to its ability to collect resource information in a larger area and therefore clients can know about more hosts in searching via rendezvous points. However, to achieve the best performance of a rendezvous point based resource discover scheme, the rendezvous point scheme needs

to be within short access distance of non-rendezvous points and load on different rendezvous points should be roughly equal to avoid overload one or a few rendezvous points. There is also an one-time rendezvous point selection cost and small maintenance cost when using rendezvous point search. In next chapter, we will introduce dynamic rendezvous point selection methods in peer-to-peer overlay networks, including our rendezvous point selection methods in structured overlay network for an even distribution throughout the overlay and balanced load on rendezvous points, and a new model that uses virtual rendezvous point for coordinating clients and volunteer hosts.

## CHAPTER 5

# Dynamic Rendezvous Point Selection in Peer-to-Peer Overlay Networks

Results in Chapter 4 Show that in a peer-based desktop grid system, the rendezvous point resource discovery method performs better than the probing-based methods such as expanding ring search, random-walk and advertisement, when considering both resource discovery effectiveness for job throughput and message-passing overhead.

In this chapter we explore the question of how to select a set of rendezvous points in a peer-to-peer overlay network to support fast discovery of available hosts by the application scheduler. We first define the rendezvous point selection problem abstractly and outline the challenges associated with rendezvous point selection in a dynamic, large scale peer-to-peer network. We investigate protocols for rendezvous point selection in unstructured overlay networks, and show that these protocols are not suitable for fast resource discovery in peer-based grids. We then develop a rendezvous point selection protocol for structured overlay networks that addresses most of the problems encountered in previous rendezvous points selection protocol. Our

key contribution is the introduction of the notion of *virtual rendezvous points* within a *resource-aware structured overlay network*. This novel concept supports low latency resource discovery combined with resilience, security, and scalability.

We broadly define the *rendezvous point selection problem* as that of selecting some subset of the peers in a large scale peer-to-peer overlay network to take a special role, with the designated rendezvous point providing service to the non-rendezvous nodes. The specially selected peers must be *well-dispersed* throughout the peer-to-peer overlay network, and must typically fulfill additional requirements such as load balance, resources, access, and fault tolerance. The rendezvous point selection problem occurs across a spectrum of peer-to-peer applications, including file sharing systems, distributed hash tables (DHTs), publish/subscribe architectures, and peer-to-peer cycle sharing systems [56, 89, 68, 46]. The rendezvous point selection problem also shows up in the fields of sensor networks, ad-hoc wireless networks, and peer-based Grid computing [68, 55, 34, 35].

It is a big challenge to design a dynamic rendezvous points selection algorithm in peer-to-peer overlay networks, which both supports low access latency from the non-rendezvous-points to the rendezvous points and adapts to dynamic, unreliable, large-scale peer-to-peer overlay networks. In the peer-to-peer environment, a large number of rendezvous points must be selected from a huge and dynamically changing network in which neither the node characteristics nor the network topology are known a priori. Thus, simple strategies such as random selection do not work, since the even distribution of rendezvous point relies on a uniformly populated topology. Rendezvous point selection protocols are challenging because they must respond to dynamic joins and leaves (churn), operate among potentially malicious nodes, and function in an environment that is highly heterogeneous.

Since the rendezvous points selection problem is closely coupled to the underlying overlay network, solutions to this problem are often tailored to the particular type of the peer-to-peer overlay networks: unstructured peer-to-peer overlay network

and structured peer-to-peer overlay network. While many researchers have proposed solutions to the rendezvous points selection problem in unstructured overlay network [56, 89, 46], few have addressed the problem of dynamic rendezvous points selection in structured overlay networks.

We designed a rendezvous point selection protocol for structured overlay networks, called SORPS (Structured Overlay Rendezvous Point Selection) [68]. SORPS takes advantage of the regular node label in a structured overlay network to create an even distribution of the rendezvous points with low message exchange and instant rendezvous point discovery latency.

SORPS is scalable and can achieve an even distributions of the rendezvous points on the overlay network. However, we found that there are no easy remedies for some of the drawbacks of SORPS such as fault-tolerance, burdens on the physical rendezvous points and security problems. To solve this problem, we developed the notion of *virtual rendezvous points* in a *resource-aware overlay network*, which achieves all the merits of a rendezvous points scheme while removing the drawback of reliance on physical rendezvous points. A virtual rendezvous point is a subspace within the structured overlay network (represented by a subset of the node labels) corresponding to a specific collection of resources. A client's request will be automatically routed to the virtual rendezvous point and then finally reach the appropriate hosts with low resource discovery overhead.

In the rest of this chapter, we will discuss the criteria of rendezvous points selection problem in peer-to-peer overlay networks. Then we will survey the rendezvous points selection protocols in unstructured overlay network. Finally we will present the SORPS protocol for structured overlay network, discuss the drawback of using physical rendezvous point, and introduce the idea of *virtual rendezvous points* in structured overlay network, in which no single physical node undertakes the role of rendezvous point.

## 5.1 Problem Definition and Related Work

We first describe the unique requirements and challenges of the rendezvous point selection problem in the context of peer-to-peer systems. We then describe dominating set and  $p$ -centers problems from graph theory related to the rendezvous point selection problem. We conclude with a survey of rendezvous point selection protocols that have been developed for unstructured overlay network.

### 5.1.1 Criteria of the Rendezvous Points Selection Problem

First, we describe key topological distribution criteria that the rendezvous points must fulfill relative to the non-rendezvous points. Second, we enumerate characteristics of peer-to-peer networks that make rendezvous point selection difficult.

**Rendezvous Point Distribution Criteria.** The rendezvous points must be distributed throughout the peer-to-peer overlay network in a topologically sensitive way to meet one or more of the distribution criteria listed below.

- *Access:* non-rendezvous points must have low latency access to one or more rendezvous points. Access can be measured in hop counts or delay.
- *Dispersal:* rendezvous points must be evenly distributed throughout the overlay network; they should not be clustered within only a few subregions of the overlay.
- *Proportion:* a pre-specified global ratio of rendezvous points to non-rendezvous points must be maintained to meet application-specific performance requirements.
- *Load balance:* rendezvous points should not serve more than  $k$  non-rendezvous points, where  $k$  can be configured locally based on the resource capability of each rendezvous point. Note that these criteria are inter-related in that specification of requirements for one may impact another, or



**Peer-to-Peer Factors.** The design of rendezvous point selection protocols within a peer-to-peer environment is challenging because in addition to fulfilling the distribution requirements outlined above, they must also deal with factors arising from the underlying nature of large scale, highly dynamic systems.

- *Heterogeneity:* Current large-scale peer-based systems consist of a large number of heterogeneous nodes with differing hardware and software resources. A node might not be eligible to be a rendezvous point unless it meets certain minimum qualifications. These qualifications include *resources*, such as CPU power, disk or memory space, or battery life; *stability*, such as uptime or fault tolerance; *communication*, such as bandwidth or fan-out; and *safety*, such as trust or security.
- *Adaptability to churn:* Peer-to-peer environments are extremely dynamic. Rendezvous points selection protocols must be able to handle churn and respond quickly, especially when rendezvous points leave the system. Rendezvous point selection protocols must also be adaptive to dynamic changes in network traffic and overlay topology.
- *Resilience and fault tolerance:* When a given rendezvous point dies, other rendezvous points should quickly take over its functions or a new rendezvous point should be quickly selected.
- *Security:* Rendezvous point may be vulnerable to denial of service attacks. Malicious rendezvous point can disrupt the system by failing to forward messages or by giving out wrong information.

### 5.1.2 Related Theoretical Problems

A wealth of research in graph theory, location theory, and distributed computing provides a formal foundation for the rendezvous point selection problem.

The basic *dominating set problem* is the problem of finding a minimal subset of the vertices in graph  $G$ , called the dominator set, such that every node is either a dominator or adjacent to a dominator. Dominating set problems and algorithms are described thoroughly in [50], and most versions are NP-hard.

*Distance domination* seeks to find a minimum size  $d$ -dominating set such that the distance from an arbitrary node to a dominator is  $\leq d$ . *Multiple  $(c,d)$ -domination* requires that every peer be within distance  $d$  of  $c$  dominators. *Colored domination* presumes that each node in graph  $G$  has an associated color from the set  $c_1, c_2, \dots, c_n$ . A dominating set of color  $c_i$  is one in which the dominators are all of that color. Colored domination can be used to model heterogeneous networks in which only certain nodes are qualified to be dominators.

The  $p$ -center problem is applicable when placing a *fixed* number of rendezvous points in a network. Algorithms and variations on this NP-hard discrete location problem are found in [47]. The  *$p$ -center problem* is the problem of finding a subset of  $p$  vertices in a graph  $G$ , called centers, to minimize the maximum (or total) distance between a non-center node and its nearest center. *Colored  $p$ -centers* can be used in a colored graph for the problem of finding a subset of  $p_i$  vertices of color  $c_i$  to minimize the above distance criteria.

The classic leader election problem from distributed computing differs from rendezvous point selection in that the former assumes all nodes vote (directly or indirectly) on the choice of each rendezvous point. Leader election algorithms are not scalable because they require broadcasting or passing a token to all nodes. The best known *leader election* protocols electing a leader (typically the node with highest ID number) under various fault tolerant scenarios, such as Ring, Bully, etc. [96, 71].

Heuristic algorithms developed for these classic problems have been utilized in the field of networking, but their applicability is usually limited to smaller scale, static networks. For the most part, they involve centralized algorithms or high message passing overhead. These algorithms were not designed for large scale peer-to-peer networks that exhibit a high degree of churn and that are dynamically heterogeneous.

### 5.1.3 Rendezvous Point Selection in Unstructured Overlay Networks

Previous research about rendezvous points selection methods are all focused on rendezvous points selection in unstructured overlay networks. The rendezvous point scheme was originally used in peer-to-peer file sharing applications. Researchers also explored the power of rendezvous points in other applications, such as security monitoring systems and peer-based desktop grid systems. *Supernode* and *ultrapeers* are used as interchangeable terms for *rendezvous points* in the literature.

The best known example of rendezvous point selection in a peer-to-peer unstructured overlay is the *gnutella* protocol [56, 89] for selection of ultrapeers—peers with sufficient bandwidth and processing power to serve as proxies for other peers. The use of ultrapeers reduces network traffic and speeds up content discovery. Figure 5.1 demonstrates the two-tier network structure of the *gnutella* peer-to-peer network.

With the *gnutella* ultrapeer selection protocol, any peer can select itself as an ultrapeer if it meets the following criteria: it has been up for at least 5 minutes, has high bandwidth, sufficient processing power, runs an OS that can handle a large number of simultaneous TCP connections, and is not firewalled/NATed. The ultrapeer selection protocol dynamically adjusts the number of rendezvous points as follows: if a new peer cannot find an ultrapeer with free slots, it can promote itself to be an ultrapeer if it is qualified to be an ultrapeer. Otherwise it does an expanding ring search to search for ultrapeers. The *gnutella* ultrapeer selection protocol is backwards

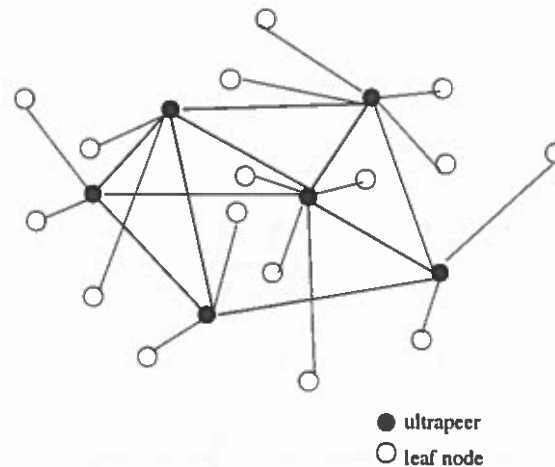


FIGURE 5.1: Gnutella two-tier hierarchical overlay network.

compatible with the original flat gnutella overlay network construction protocol without ultrapeers, therefore if peer can still join the overlay network if no ultrapeers are found by adding a connection to the contacted peer.

Gnutella's ultrapeer selection protocol adapts dynamically to users' needs in a best effort fashion by limiting the number of leaf nodes one ultrapeer can handle and adding ultrapeers when needed. Therefore, the gnutella protocol loosely meets the load balance criteria. With the localized expanding ring search and self-promotion when no ultrapeers are found, it also loosely meets access and dispersal criteria. However, gnutella cannot fulfill the proportion criteria. Even though one of gnutella's goals is to achieve a certain ratio of ultrapeers to leaf nodes, currently there is no way to control this ratio.

The  $H_2O$  (Hierarchical 2-level Overlay) protocol [68] for rendezvous point selection is a distributed negotiation protocol for unstructured overlay networks used in the Sequoia [55] security monitoring system.  $H_2O$  uses a classic advertisement-based protocol, in which rendezvous points advertise security-related resource information, and non-rendezvous points cache these advertisements. Non-rendezvous points can then choose to join the best rendezvous point(s) using locally cached information. This

protocol gives full autonomy to both rendezvous points and non-rendezvous points, allowing each to negotiate using its own local policy. *H<sub>2</sub>O* is similar in many ways to the gnutella protocol, but allows for finer-grained control over the rendezvous point selection process e.g., it can consider trust, secure paths, and routing performance. The *H<sub>2</sub>O* protocol includes three steps: rendezvous point advertisement, rendezvous point search and final handshake.

J. Han [46] proposed to use a gnutella-like rendezvous point based scheme to build a desktop grid system in an unstructured overlay network. The paper proposed to group peers into clusters without clearly defining how peers were clustered. Then the most powerful and long lived peers in a cluster will be selected as rendezvous points in that cluster. The resource information is submitted to the local rendezvous point, and queries are propagated among rendezvous points and handled by rendezvous points with available resources in its cluster. The paper is focused on usage of rendezvous points without proposing an effective rendezvous point selection algorithm.

All these rendezvous point selection methods are designed for unstructured overlay networks, which require probing based rendezvous point discovery methods with high message overhead. A newly joined peer has to send out query messages to its neighbors to acquire a list of rendezvous points; as a result each peer can only know some of the rendezvous points in the system. These protocols are more effective in adding new rendezvous points when there are more peers joining the system, but are less effective in reducing the number of rendezvous points or adjusting the position of rendezvous points when peers leave. Therefore the distribution of the rendezvous points can be very uneven, which causes uneven load on rendezvous points. To achieve fast turnaround scheduling in peer-based desktop grid system, the clients need to be able to locate the rendezvous point quickly and communicate with the rendezvous point with low latency. The load on rendezvous points should not be excessively high

so that rendezvous points will not become the bottleneck of the system. Therefore the rendezvous point selection protocols in unstructured overlay networks do not satisfy the requirements of fast turnaround scheduling.

## 5.2 Rendezvous Points Selection in Structured Overlay Networks

As we have described, existing rendezvous point selection schemes in unstructured overlay network cannot distribute the rendezvous point evenly throughout the overlay network, or balance the load on rendezvous points when peers dynamically leave and join. In addition, a client in an unstructured overlay network can only know a few rendezvous points. Rendezvous point selection protocols in structured overlay networks can circumvent the problem of high message overhead and uneven distribution of the rendezvous points by taking advantage of the node labeling scheme in the underlying structured overlay network.

### 5.2.1 SORPS: Structured Overlay Rendezvous Point Selection

A DHT (Distributed Hash Table) built on a structured overlay network such as CAN [78], Chord [93], and Pastry [83] makes use of a symmetric, regular node label space, in which each physical node owns a virtual subspace in the overlay. In these structured overlay networks, a compact *node label expression* can encode a (large) collection of virtual nodes.

We have designed the SORPS (Structured Overlay Rendezvous Point Selection) protocol to exploit this notation and uses a node label expression to designate a subset of the virtual node label space as rendezvous points. The rendezvous point label expression is stored in the DHT for fast and easy lookup. The number of

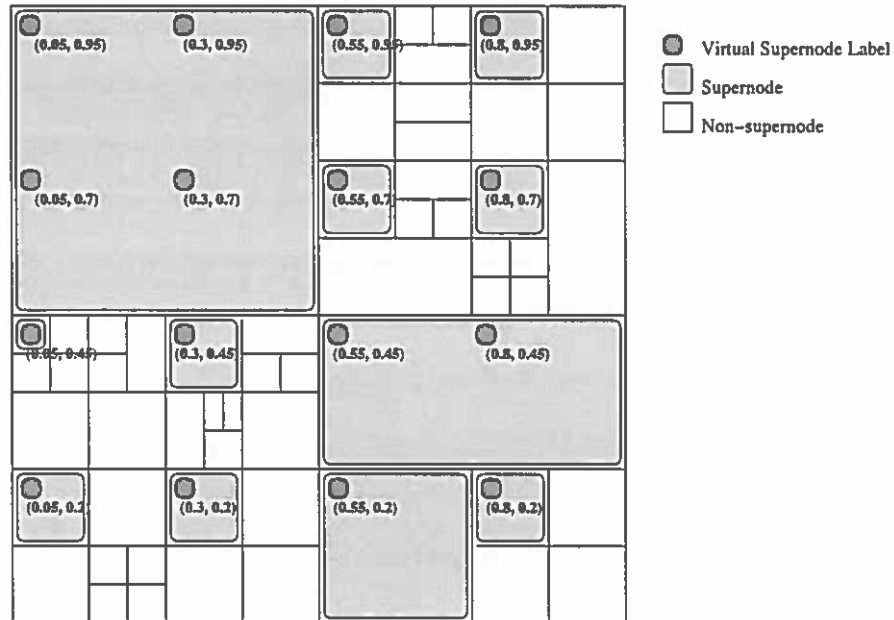
rendezvous points can be expanded simply by changing the node label expression (see examples below). Because a structured overlay network maps physical nodes to virtual subspaces in a manner that is sensitive to both density and topology, the rendezvous points are evenly distributed among physical ordinary peer nodes and every ordinary peer has one or more nearby rendezvous points.

SORPS can keep the rendezvous point to ordinary peer ratio stable as peers join and leave the overlay. It also maintains low access from ordinary peers to rendezvous points and provides load balancing for each rendezvous point. In the last part of this section, we discuss how SORPS addresses resource heterogeneity and how SORPS copes with the departure or failure of rendezvous points.

### SORPS Protocol

- **Initiation of SORPS rendezvous point selection.** A node that wishes to initiate the rendezvous point selection procedure for some service hashes information about the service into the DHT. This includes the public key of the initiator and the rendezvous point selection policy. This policy contains the rendezvous point label expression, the minimum criteria for a node to be a rendezvous point, and maximum lifetime of a rendezvous point. An ordinary peer can discover the identity of rendezvous points for this service by accessing the DHT using the service related key to lookup the rendezvous point label expression.

*Example 1: CAN structured overlay.* Assume the CAN space is a  $d$ -dimensional space and the length of the  $i_{th}$  dimension is  $d_i$ . To select  $k$  rendezvous points, the service initiator first factors  $k$  into  $k_1, k_2, k_3, \dots, k_d$  where  $k = \prod_{i=1}^d k_i$ . The  $i_{th}$  dimension of the rendezvous point label should obey the formula  $(b_i + n * d_i / k_i) \% d_i$ , where  $b_i$  is a random number chosen in the range of  $[0, d_i]$  and  $n$  is an integer. The randomness prevents different service initiators from choosing the same group of rendezvous points. Figure 5.2 illustrates this process for 16 rendezvous points in a two-dimensional CAN space.



**FIGURE 5.2:** Rendezvous point selection in CAN. (Rendezvous point label expression is  $(0.05+0.25 n)\%1,(0.20+0.25 m)\%1$ )

*Example 2: Pastry structured overlay.* To select  $k$  rendezvous points, the service initiator needs to generate a node label with  $\lceil \log_2 k \rceil$  don't-care bits as the highest order bits (or the don't-care bits can be distributed randomly within the label); the remaining bits in the node label are randomly set to 0 or 1. For example, if the service initiator needs 1024 rendezvous points it will use  $\underbrace{\times \times \times \cdots \times}_{10 \text{ bits}} \underbrace{1001 \dots 1}_{118 \text{ bits}}$  as the rendezvous point label expression. Any node whose label matches the 118 instantiated bits is a potential rendezvous point. When a message are sent towards the rendezvous point label, the physical node with the closest node label will receive it.

- **Rendezvous point takes charge of the service.** The initiator can send a message towards the rendezvous point labels to inform the chosen rendezvous points. The notification is done via multicast in the overlay network. Each physical node that owns a node whose label matches the rendezvous point label expression will receive the message and become a rendezvous point. Alternatively, a rendezvous point takes charge upon receiving the first request from an ordinary peer.



- **Ordinary peer joins service.** If an ordinary peer wants to join a service, it looks up rendezvous point label expression in the DHT. The ordinary peer can then figure out the nearest rendezvous point in the virtual overlay according to the routing protocol. The structured overlay network provides bounded virtual routing with path length proportional to the distance between labels in the label space. In CAN the ordinary peer uses the Cartesian distance between its own label and the rendezvous point label to estimate distance. In Pastry, the ordinary peer uses the bit-wise XOR operation to compute the label distance from which it estimates the physical distance.

### SORPS Features

We have also designed the following extensions to SORPS to fulfill the additional requirements of the dynamic heterogeneous peer-to-peer overlay networks:

- *Resilience and fault tolerance.* SORPS must function in situations in which rendezvous points depart gracefully or die suddenly. A rendezvous point can replicate rendezvous point-related state on the neighbor nodes that will take over the subspace covered by its label if it fails. Rendezvous point failure will be detected by the underlying overlay network maintenance protocol. The most capable neighbor of the failed rendezvous point can then take over the rendezvous point role.
- *Heterogeneity.* Rendezvous points should be those nodes with better capability with respect to CPU speed, network connections, and other resources. If a new node joins towards an existing rendezvous point coordinate, the new node and the existing physical node serving as a rendezvous point can negotiate to see which one is more capable to take the rendezvous point role. Finally, a nearby ordinary peer can offer to take over a nearby rendezvous point's role if it is more capable by swapping virtual subspaces.

- *Adaptability.* An ordinary peer can switch to another rendezvous point when it is not satisfied with the performance of its current rendezvous point. It can determine the distance to other rendezvous points by comparing its own node label expression with the rendezvous point's node label expression, or it can query the DHT to see if new rendezvous points have been selected by the initiator. When more rendezvous points are needed the initiator simply expands the rendezvous point label expression to cover more virtual node labels.

### SORPS for Scheduling in Peer-based Desktop Grid

Rendezvous point based scheduling uses the SORPS protocol described above to select a group of rendezvous points in the system. Rendezvous points function as matchmaking servers in the system, similar to the matchmaking servers in Condor [77].

- **Hosts report status.** When a host becomes available, it sends a status report to register with its nearest rendezvous points, reporting its CPU clock rate, memory size and operating system. The rendezvous point will cache the status report of the host.
- **Clients send request.** When a client wants to find available hosts to compute its job, it sends a request to the rendezvous point including requirements about CPU clock rate and memory size.
- **Rendezvous point matches the request with available hosts.** On receiving the request, the rendezvous point consults its cache and finds the matching hosts. The rendezvous point then sends the list of candidate hosts to the client for it to select the best host to use. If it cannot find any matching hosts, the

rendezvous point can forward the query to another rendezvous point. Alternatively, it can return a negative report to the client, and the client can contact another rendezvous point.

- **Clients choose the best hosts.** The client chooses the best host according to its local policy and contacts the host. A typical local policy for fast turnaround can include choosing the host with the highest clock rate and choosing the host, which recently became idle. If the host is still available and decides to accept the job based on its local scheduling policy, it will send a message to the nearest rendezvous point to cancel the registration.
- **Hosts become unavailable.** When a host becomes unavailable, the foreign job can sleep on that host or must be preempt according to the local policy of the host. When the foreign job is preempted, the client has to reschedule the job. If checkpointing is available, the client can migrate and restart the job on another host.

### Analysis of SORPS's Performance

As SORPS uses structured node labeling schemes to form the rendezvous point node label expression, it can provide a theoretical upper bound for several key rendezvous point selection criteria: *proportion*, *load balance* and *access*. In this section, we analyze these theoretical upper bound for SORPS in the Pastry structured overlay network. The performance of SORPS in other structured overlay networks can be analyzed using a similar approach.

The Pastry network uses prefix-based routing based on node label and it uses a 128-bit binary node label. In the routing table, node labels are interpreted as unsigned integers in base  $2^b$  (where  $b$  is a parameter with typical value 4). We assume The

rendezvous point node label expression in Pastry represents a group of rendezvous points using do-not-care bits. We assume that the number of do-not-care bits in the rendezvous point node expression is  $n$  in the following analysis, and the number of nodes in the Pastry network is  $N$  ( $O(2^{128})$ ).

- **Proportion.** We measure the proportion criteria by the ratio of rendezvous points to non-rendezvous points. The number of rendezvous points is  $O(2^n)$ . Therefore the ratio of rendezvous points to non-rendezvous points is  $O(2^n/N)$ .
- **Load balance.** We measure load balance by maximal number of non-rendezvous points handled by one rendezvous point. In best case that that node are evenly distributed in the virtual node label space including the rendezvous point, the number of virtual non-rendezvous points handled by one physical rendezvous point will be  $O(N/2^n)$ . The even distribution of nodes depends on the hash function used by Pastry to generate node labels.
- **Access.** We measure access by latency between the non-rendezvous points to its nearest rendezvous point. The virtual rendezvous points differ in  $n$  bits. Therefore routing to the nearest non-rendezvous points is the same as routing in a Pastry network with node label space of  $(128 - n)$  bits. Using the same type of the original mathematics analysis in original Pastry paper, routing in this Pastry network is bounded by  $O(\log_{2^b} 2^{128-n})$ , in which  $2^b$  is the base used in interpreting the numerical node label.

The performance of SORPS in Chord or Tapestry is similar as they use the same binary node label and similar node label based routing. In CAN overlay networks, the latency between one non-rendezvous point to its nearest rendezvous point is  $O(n^{1/d}/K)$ , as length of routing path in CAN is bounded by  $O(n^{1/d})$ .

### 5.2.2 Drawbacks of Physical Rendezvous Points

The rendezvous points selection schemes we have discussed have a common drawback: a few physical nodes carry a lot of the burden. These physical nodes are in charge of collecting resource information and matching clients' requests with available hosts. Therefore, there are high requirements on availability and capability, such as network connection speed and CPU speed of the rendezvous points. It is possible that qualified physical nodes do not even exist in parts of the overlay network. In addition, the peers are potentially the target of attacks, and even worse, potentially malicious. Dependence on the reliability and trustworthiness of those physical rendezvous points opens performance and security holes in the system.

While the fault-tolerance of the system can be improved via replication, it is often complicated and involves extra overhead. A host can send resource information to more than one rendezvous point, and a client can also send its request to more than one rendezvous point. However, this scheme causes higher message overhead and higher burden on those rendezvous points. A rendezvous point can replicate its information on its neighbors, hoping that one of its neighbors can take over its function when it fails. However, periodic updates of the resource information on the neighbors also introduces high message overhead, especially when the resources on the hosts are highly dynamic.

Securing these type of rendezvous points selection schemes is a much more serious problem than providing fault-tolerance. In this asymmetric system, rendezvous points take a much more powerful position and possess more knowledge about the system than the ordinary peers. A malicious rendezvous point can attack the system and stop it from functioning with the additional knowledge and privileges it has. For example, it can keep forwarding requests to busy hosts, or refuse clients' requests even when there are available hosts. It is hard to detect this type of attack, as the operations on rendezvous points are not transparent to the ordinary peers. When the resource information is sent and replicated to multiple rendezvous points, incon-

sistency in the response to clients' requests by multiple rendezvous points may alert the ordinary peers about potential attacks, but some harm may be already done to the system. Several malicious rendezvous points may collude in propagating faked resource information. The scenario of colluding malicious peers adds another layer of complication to the solution of the whole problem, as it is even harder to detect the attack. Furthermore, it is hard to prevent malicious nodes from occupying some rendezvous points position, since the rendezvous point selection algorithm is known to all peers. A malicious node can always disguise itself as a qualified node and occupy a strategic position in the overlay network.

Finally, there needs to be a motivation for peers to function as rendezvous points since rendezvous points contribute more to the system and it is natural that they would expect more rewards in return. The success of the gnutella [56] and kazaa [85] peer-to-peer file sharing system shows that the extra burden of being rendezvous points (supernodes) is acceptable to peers in this type of systems. It is unclear whether this observation will still hold in a peer-based desktop grid.

### 5.3 Virtual Rendezvous Points Schemes

As discussed above, a physical rendezvous point scheme has inherent drawbacks including high burden and reliance on physical rendezvous points, as well as worse fault-tolerance and security problems. The problem boils down to whether we can design a system which sustains the benefits of a rendezvous point based system but without the drawbacks. This type of system should serve the same essential role as a rendezvous point scheme: the resource information of hosts in a larger area than the local neighborhood is available to all clients. However, this type of system should not rely on a few physical nodes in the system, and it should not make some physical nodes in a position superior to the other nodes. Finally, from the a performance standpoint, the overhead of selection and maintenance of the rendezvous points needs to be low.

We introduce the idea of *virtual rendezvous points* in a structured overlay network, which does not require any physical node to take charge of the responsibilities of a rendezvous point. Instead, resource information is built into the structure of a *resource-aware overlay network* which encodes resource information using the node labeling scheme associated with a given structured overlay network such as CAN or Pastry.

A *RAON* (*resource-aware overlay network*) is a structured overlay network which uses its native node labeling scheme to encode resource information. The types of resource within each resource class must be finite (enumerable) and stable.

A *virtual rendezvous point* is a subspace within the structured overlay network (represented by a subset of the node labels) corresponding to a specific collection of resources.

Virtual rendezvous points occur in the RAON as follows: hosts join the RAON by placing themselves at a coordinate in the subspace that represents their resource capabilities. Clients discover resources by routing messages directly to a coordinate in the subspace that represents their resource needs.

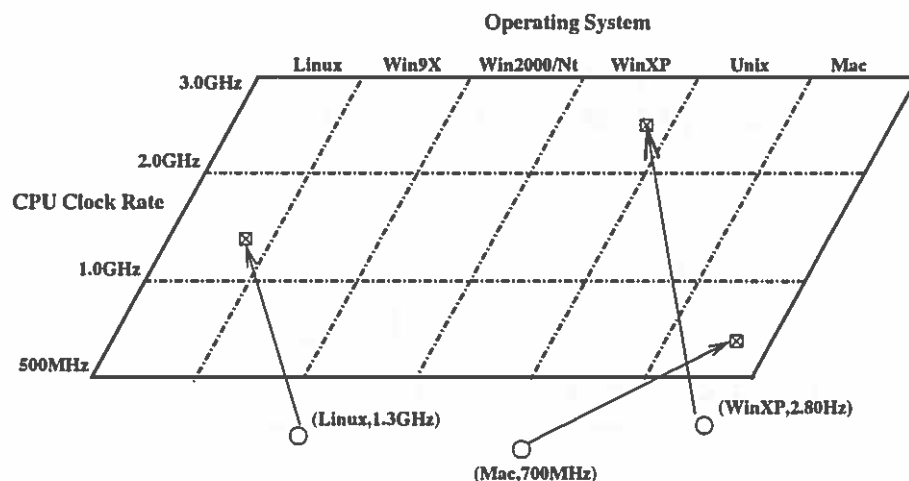
When a client queries for hosts satisfying its requirements, its request will be automatically routed to the matching hosts by the underlying DHT overlay routing protocol. A client can collect a group of hosts as the node label expression used for routing may encode a group of hosts, and the client can then choose the best from the group of candidates.

When the resources on a host changes, the host needs to leave the overlay network and joins a new location matching its new resource specification.

### 5.3.1 Examples of RAON

Figure 5.3 shows an example of RAON, which is a 2-dimension RAON providing both operating system and CPU clock rate information. One dimension in this RAON represents types of operating system, and the other dimension represents CPU clock

rate. There are 18 virtual zones (virtual rendezvous points) in the overlay network. Hosts will join a particular zone in the overlay network, according to its operating system and CPU speed. For example, a host running on WinXP with a 2.8GHz CPU clock rate will join a random location in the zone containing all hosts running on WinXP and with a CPU clock rate in the range of 2.0GHz to 3.0GHz. Notice that the figure shows an idealized even distribution of hosts in the resource description space. A realistic ROAN would be designed to accommodate an uneven heterogeneous distribution of hosts. In addition, the granularity of the resource profile can be improved by further dividing the zones. For example, the dimension representing types of operating system can be further divided according to different versions and different service package. Different type of clients' requests can be expressed by different node label expression. If a client wants to search for a Linux machine with CPU speed in the range of 1.0GHz to 2.0GHz, it can send a request (Linux,  $k$ ) which  $k$  is a value greater than 1.0 and less than 2.0. However, if it does not care about CPU speed, it can send a request (Linux, 'X') using a do-not-care 'X' in the position representing CPU speed.



**FIGURE 5.3:** An example of ROAN, the information of operating system and CPU clock rate on hosts is embedded into the overlay network.



The binning scheme used by the topology-aware CAN overlay network [79] is a special case of RAON. When a new node joins, it pings the well-known landmark nodes and generates a landmark label according to the order of the landmarks based on its distance to them. The node then joins the other nodes with the most similar landmark label in the CAN space. Virtual rendezvous points are formed according to the number and locations of the landmarks, and a host registers with the right bins according to its network location.

### 5.3.2 Advantages and Limitation of RAON

There are many advantages of virtual rendezvous points compared with physical rendezvous point scheme such as fast resource discovery speed, minimal search overhead, light-weight and secure architecture.

- **Fast resource discovery with minimal search overhead.** According to the underlying node labeling scheme, the clients' requests will be automatically routed to matching hosts. DHT routing provides fast resource discovery without the delays and overhead associated with probing-based resource discovery and disturbing unrelated nodes.
- **Light weight.** ROAN is a light weight system in which no physical host stores resource information. Instead, the resource information is embedded into the topology of the overlay network. It reduces the burden on physical nodes, as no physical nodes take charge of storing the hosts' resource information. In addition, there is no extra overhead to distribute, store, and maintain the resource information.
- **Fault-tolerance.** In contrast to physical rendezvous point, ROAN does not need the heavy-weight replication-based fault-tolerance, as the resource information is not stored on a few physical nodes. When a host leaves or fails, only

the resources on that host are lost and resource information about other hosts are still available as they are embedded into the topology of the overlay.

- **Better security.** There is no physical vulnerable point in ROAN, as no physical node stores the resource information of the hosts. The attackers cannot start an attack aimed at compromising the rendezvous points and ultimately the whole system.

The only drawback of RAON is that the type of resources it can represent is limited. ROAN is efficient in dealing with stable resources, however it is less efficient in handling dynamic resource information such as files. RAON cannot efficiently represent content location, as the files can be duplicated and can be deleted. Each redistribution and deletion of the contents requires hosts involved in the actions to rejoin the overlay network. This will cause high maintenance overhead and severe churn in the network. That is why content location is handled by additional file keys other than node label alone in DHT overlay networks designed for peer-to-peer content sharing.

### 5.3.3 Timezone aware RAON for Peer-based Desktop Grids

So far our discussion of RAON has been limited to static host resources such as operating system type and CPU power. The most critical resource that must be specified for cycle sharing systems is blocks of idle CPU time on the hosts. Studies have consistently shown that large relatively stable blocks of idle time are available on desktop machines, most notably night-time cycles on home machines or office workstations. This information can be build into a RAON for fast resource discovery by clients seeking idle cycles by construction of a timezone-aware RAON in which the node label space is divided into international timezones based on the GMT system.

Figure 5.4 shows a timezone-aware overlay network for the CAN structured overlay network with 24 virtual rendezvous point zones. For simplicity, only a few virtual zones corresponding to timezones are shown in the graph. The hosts choose a random label in the particular virtual zone corresponding to the timezone it is in, and then join that virtual zone. A host in Portland, USA will join zone [5:00,6:00] and a host in Beijing, China will join zone [20:00,21:00]. Using this time-zone aware overlay network, it is easy to locate hosts at night. For example, a client submitting a job at 8:00am pacific time may schedule it in virtual zone [20:00,21:00], as hosts in this zone are in midnight and have a high probability to be idle.

Note that a timezone aware overlay network can be designed with larger granularity and additional resource information can be encoded in other dimensions of the overlay.

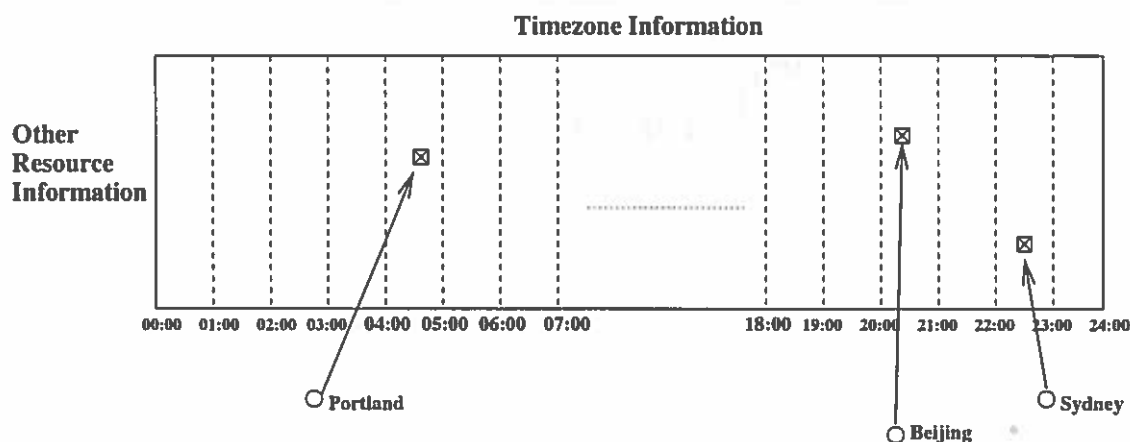


FIGURE 5.4: An example of Timezone-aware RAON. Hosts join according to their timezone information.

### 5.3.4 Summary

In the chapter, we have described the SORPS rendezvous point selection method in structured overlay networks, which has many advantages over the rendezvous point selection methods in unstructured overlay network. These advantages include con-

**TABLE 5.1:** Comparison of SORPS with rendezvous point selection in unstructured overlay network (RP: rendezvous point)

	<b>SORPS</b>	<b>Rendezvous Point Selection in Unstructured Overlay</b>
<b>Search for RP</b>	Node label based	Probing based
<b>Message Overhead</b>	Low	High
<b># of RP Known by Non-RP</b>	All	A few
<b>Proportion</b>	Bounded ratio	Unbounded Ratio
<b>Access</b>	Bounded latency	Unbounded latency
<b>Load Balance</b>	Balanced	Unbalanced
<b>Fault-tolerance</b>	Good	Better
<b>Complexity</b>	Complex	Simple
<b>Maintenance Cost</b>	High	Low

stant rendezvous point discovery time and even distribution of the hosts. Table 5.1 summarizes the advantage of SORPS compared with the rendezvous point selection schemes in unstructured overlay networks. However, SORPS has some inherent drawbacks regarding fault-tolerance and security. We developed the notion of virtual rendezvous points which provides the function a rendezvous point system but does not have the disadvantage of a physical rendezvous point scheme. Table 5.2 compares the physical rendezvous point scheme with the virtual rendezvous point scheme.

**TABLE 5.2:** Comparison of physical rendezvous point with virtual rendezvous points (RP:rendezvous point)

	<b>Physical RP</b>	<b>Virtual RP</b>
<b>Client query</b>	Contact RP Matchmaking	Auto dissemination using DHT routing
<b>Host resource profile</b>	Deposit on RP	Embedded in the overlay
<b>Burden on Physical nodes</b>	Yes	No
<b>Fault-tolerance</b>	Replication-based	Inherent
<b>Security concern</b>	RP vulnerable to attach Malicious RP	No security concern related to physical RP
<b>Resource represented</b>	arbitrary	stable

Our analysis led us to conclude that RAON holds the most promise for fast resource discovery in a peer-based desktop grid. It combines all the advantages of a rendezvous point approach with very few disadvantages. In the next chapter, we discuss the design of a timezone-aware RAON which we call WaveGrid for use in heterogeneous peer-based grids. We use WaveGrid as the infrastructure for the next stages of our research into fast turnaround scheduling.

## CHAPTER 6

# WaveGrid: Scheduling for Fast Turnaround in Open Peer-based Desktop Grid Systems

### 6.1 Introduction

In chapter 5, we have introduced the concept of resource aware overlay network(RAON), which supports scalable, fast and efficient resource discovery in a structured peer-to-peer overlay network. In this section, we show a new scheduling method for fast turnaround in open peer-based desktop grid systems combining RAON techniques and migration.

Although peer-based desktop grid systems are lightweight, easy to assemble and operate, and do not suffer from bottlenecks at the central servers, there are problems, such as fast turnaround scheduling, scalable resource discovery, incentives and security. In this study, we focus on the problem of efficient scheduling for fast turnaround in a peer-to-peer environment.

The challenges to design a peer-based desktop grid system which satisfies jobs with fast turnaround requirements are rooted in the nature of this type of system.

First, it is difficult to collect accurate global resource information in a large dynamic peer-based desktop grid system, as collecting resource information on all the nodes is unscalable and resources are dynamically changing. Smart resource discovery methods need to be designed, which require minimal message exchange but provide sufficient resource information for the scheduler to make good scheduling decisions.

Second, the resources in a peer-based desktop grid are volatile. Peers may join and leave. Resource owners may withdraw their resources at any time. Schedulers in peer-based desktop grid systems are faced with frequent changes in this dynamic environment and need to make fast adjustment accordingly.

Third, in this opportunistic system, local jobs should have higher priority than foreign jobs. As a result, the foreign job will make slower progress since it can only access a fraction of the host's CPU availability. Schedulers must make use of as many as idle cycles as possible to satisfy the need of the foreign jobs for fast turnaround.

Fourth, the system is heterogeneous, as hosts have different CPU clock rate, different memory size and run on different operating systems. Scheduling strategies must take the heterogeneity of the hosts into consideration.

Finally, each node is an autonomous system, so scheduling in peer-based desktop grid system must be non-intrusive. Scheduling methods relying on heavy performance monitoring are inappropriate as users, especially home machine cycle donors, will find it is intrusive to report their CPU usage periodically to some remote clients.

Our solution to the problem, *WaveGrid*, is a novel scalable heterogeneous peer-based desktop grid system for fast turnaround [104, 105]. Different from previous peer-based desktop Grid systems, *WaveGrid* is motivated by the natural distribution pattern of idle cycles: users have daily routines in using their machines. The most noticeable idle cycles are the night-time idle cycles, and day-time idle cycles of home machines. In addition, hosts are geographically distributed in different timezones

on the Internet, and the area which contains the most idle hosts changes over time. Our contribution resides in two novel components to form WaveGrid: a *self-organized timezone-aware overlay network* and an *efficient scheduler using migration*, designed to take full advantage of this characteristic of the idle cycles.

- **Self-organized timezone-aware overlay network.** The timezone-aware overlay network used by WaveGrid is a special case of ROAN. Virtual rendezvous points are built based on timezones, as hosts self-organize by timezone to indicate when they have large blocks of idle time.

As in ROAN, the timezone-aware overlay network provides a mechanism for clients to find idle hosts without the high overhead of traditional blind-search-based research discovery strategies [53, 103]. It takes constant time for the scheduler to choose the targeted area to search for an available hosts based on the node-label, then a limited scope expanding ring search is conducted in that area to discover a group of candidates. For example, a host in Pacific Time timezone can join the corresponding area in the overlay network to indicate that with high probability his machine will be idle from 8:00-14:00 GMT when he sleeps.

- **Efficient scheduling and migration.** Under WaveGrid, a client initially schedules its job on a host in the current night-time zone. When the host machine is no longer idle, the job is migrated to a new night-time zone. Thus, jobs ride a wave of idle cycles around the world to reduce turnaround time.

WaveGrid is the first desktop grid system to explore the power of migration strategies for fast turnaround. When migrating, WaveGrid selects the host with the highest performance potential. In this study, we focus on CPU speed, but our model is easily generalized to other criteria.

We also propose an eager migration scheme to augment the basic migration scheme in a heterogeneous environment. Hosts in the background search for



migration target with better CPU power, achieving better performance at with slightly higher resource discovery costs.

Another contribution of this dissertation is an empirical heterogeneous host profile model for evaluating the performance of Internet-wide desktop grid systems. None of the previous desktop grid systems have used heterogeneous profiles characterizing a large number of hosts on the Internet. They either do not consider heterogeneity of the hosts or use profiles of a small number of lab or office machines. We use statistical data from the BOINC project to generate the host profile. Based on this model, we have studied the performance of WaveGrid compared with systems using a range of non-timezone-based migration strategies. The simulation results show that:

- WaveGrid outperforms other systems with respect to turnaround, stability and minimal impacts on hosts.
- WaveGrid reduces the migration delay and minimizes rescheduling attempts.
- All systems benefit from scheduling strategies that take host heterogeneity into account.

## 6.2 Timezone-aware Overlay Network and Scheduling in WaveGrid

The design of WaveGrid springs naturally from the observation that millions of machines are idle for large chunks of time. For example, most home and office machines lie idle at night. It is also influenced by the notion of prime time v. non-prime time scheduling regimes used by parallel job schedulers [66], which schedules long jobs at night to improve turnaround time.

There are many motivations for the design of WaveGrid. First, resource information, such as when the host will be idle and how long the host will continue to be idle with high probability, will help the scheduler make much better decisions. WaveGrid builds this information into the overlay network by having hosts organize themselves into the overlay network according to their timezones. Second, efficient use of large and relatively stable chunks of idle cycles provides the best performance, in contrast to using sporadic short periods of idle cycles which may be countered by high resource discovery and scheduling overhead. Therefore, WaveGrid proposes to use long idle night-time cycles. Third, the cycle donors are geographically distributed, so that their idle times are well dispersed on the human 24-hour time scale. Machines enter night-time in the order of the timezones around the world, making it well-suited for efficient migration. Fourth, the scheduler should not be intrusive to users' privacy. WaveGrid only needs minimal user input such as timezone information and CPU clock rate.

WaveGrid builds a timezone-aware, structured overlay network and it migrates jobs from busy hosts to idle hosts. WaveGrid can utilize any structured overlay network such as CAN [78], Pastry [83], and Chord [93]. The algorithm we present here uses a CAN overlay [78] to organize nodes located in different timezones and migration of jobs happens when the current host is no longer available (see Figure 6.1).

The key contribution of the timezone-aware overlay is its creation of virtual rendezvous points which provides (a) fast resource discovery without any search overhead and (b) a light weight system in which no physical host stores resource information. Hosts simply join the overlay network according to their idle timezone and clients simply hash to a host in the desired nightzone using the underlying DHT. Our work is distinct from all other DHT-based resource discovery systems [7, 51, 48, 52, 72] which query resource information stored in the DHT. These query-based systems incur overhead to cache and maintain soft-state resource profiles of the hosts.

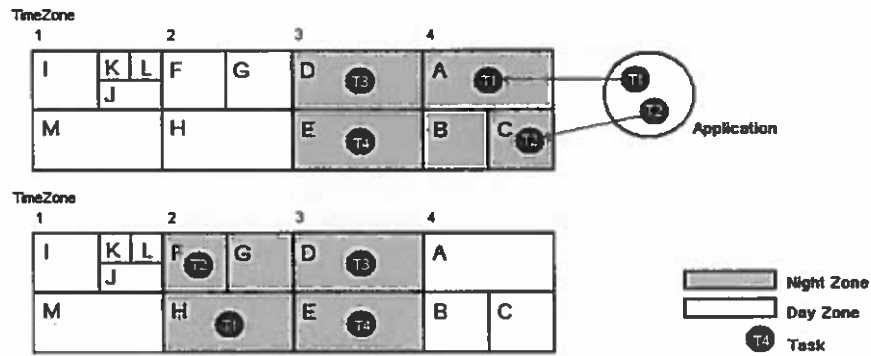


FIGURE 6.1: Job initiation and migration in WaveGrid

**Wavezones in the CAN overlay:** We divide the CAN virtual overlay space into several *wavezones*. Each *wavezone* represents several geographical timezones. A straightforward way to divide the CAN space is to select one dimension of the  $d$ -dimensional Cartesian space used by CAN and divide the space into several wavezones along that dimension. For example, a  $1 \times 24$  CAN space could be divided into 4 wavezones each containing 6 continuous timezones. Adjustments will be made for timezones with low user population.

**Host nodes join the overlay:** A host node that wishes to offer its night-time cycles knows which timezone it occupies, say timezone 8. It randomly selects a node label in wavezone 2 containing timezone 8 such as (0.37, 7.12) and sends a join message to that node. According to the CAN protocol, the message will reach the physical node in charge of CAN node (0.37, 7.12) who will split the portion of the CAN space it owns, giving part of it to the new host node.

**Client selects initial nightzone:** The scheduler for a workpile application knows which timezones are currently nightzones. It selects one of these nightzones (based on some selection criteria) and decides on the number  $h$  of hosts it would like to target.

There are a variety of nightzone selection criteria for selecting the initial wavezone and the wavezone to migrate to, including (a) schedule the task to the wavezone whose earliest timezone just entered night-time, (b) schedule the task to a random night-time zone, and (c) schedule the task to a wavezone that currently contains the most night-time zones. The first option performs better than the others, since it provides the maximal length of night-time cycles. However, it may be better to randomly select a nightzone if many jobs simultaneously require scheduling to avoid collisions.

**Host Discovery:** The scheduler randomly generates  $h$  node labels in the wavezone containing the target nightzone and sends request messages to the target node labels using CAN routing. Each contacted host does an expanding ring search in a limited scope to discover more candidates.

**Host Selection:** The application scheduler chooses the best host from the candidate group to schedule the foreign job on. The primary selection criteria in a desktop grid system is the availability of the host.

To address the concern that foreign jobs will disturb a user's local work, WaveGrid uses a strict host availability model, where CPU cycle sharing is limited to the time when owners are away from their machines and the CPU load from local applications is light. Figure 6.2 illustrates a sample host profile of available idle cycles under a strict user local policy in WaveGrid: The host is available only when the CPU load is less than 75% and there is no mouse or keyboard activity for 15 minutes. In reality, many cycle sharing systems use a conservative CPU availability model. Condor supports strict owner policies: users can specify a minimum CPU load threshold for cycle sharing, or specify specific time slots when foreign jobs are allowed on that host. SETI@home uses a screensaver model: it runs when no mouse or keyboard activities have been detected for a pre-configured time; otherwise it sleeps.

Secondary selection criteria includes the CPU power, memory size and type of operating system, etc. In this study, we focus on CPU power which is directly related to the execution time of the foreign job. When a group of candidates is selected based on CPU availability, the host with the best CPU power is chosen.

After negotiations, the application scheduler ships code to the chosen hosts.

**Migration to next timezone:** When morning comes to a host node and the host is no longer available, it selects a new target nightzone, randomly selects a host node in that nightzone, and after negotiating with that host, migrates the unfinished job to the new host.

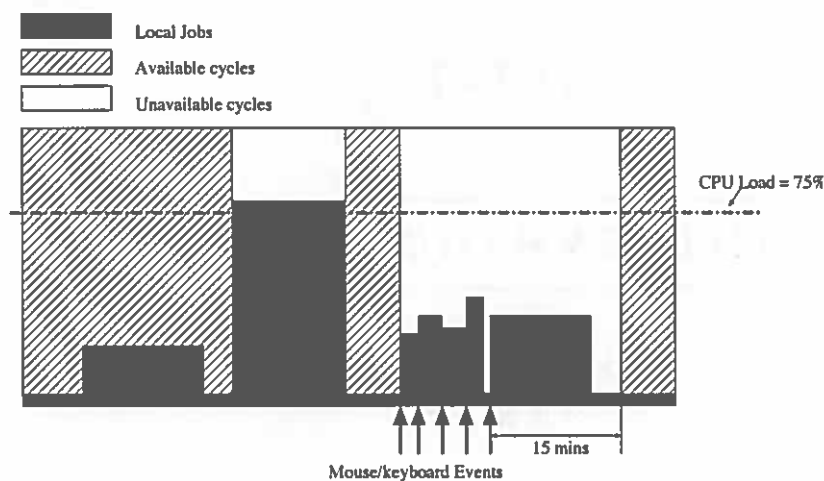


FIGURE 6.2: Sample host profile of available idle cycles

### 6.3 Migration

Migration was originally designed for load sharing in distributed computing to move active processes from a heavily loaded machine to a lightly loaded machine. Theoretical and experimental studies have shown that migration can be used to improve turnaround time [31, 88]. We believe that application of migration scheme can

also improve the turnaround in peer-based desktop grid systems, however there are often research questions to be solved such as selection of migration target and decision of when to migrate. We proposed and evaluate a variety of migration strategies with respects to overhead and success rate. In addition, with the longer transmission delay in the Internet, suitable applications needs to be identified for migration.

### 6.3.1 Suitable Applications for Migration in Peer-based Desktop Grid Systems

The type of applications which are suitable to schedule and migrate in WaveGrid are large *Workpile (Bag-of-tasks)* jobs. Each job consists of a number of independent tasks requiring large amounts of CPU cycles but little if any data communication. Examples of workpile applications include state-space search algorithms, ray-tracing programs, gene sequencing and long-running simulations. Often these applications have higher performance requirements such as faster turnaround time and higher throughput . Some of the above applications may have real time constraints, such as weather forecasting and medical diagnosis for a patient, or scientific simulations that must be completed in time for a scheduled reporting deadline.

The migration cost is higher in global peer-based cycle sharing systems than in local area networks because the code and data are transferred on the Internet. If a short running job is migrated many times in its life span, the accumulated migration cost may well counter the migration benefit. Long jobs which run for hours or even for months receive maximal benefit from migration. For such jobs, the cost of migration, which includes resource discovery overhead to find a migration target, checkpointing, and cost to transfer the code and data is negligible compared to the total runtime.

Many long running applications satisfy the migration criteria: small size and minimal data communication. For example, the average data moved per CPU hour by users of SETI@home is only 21.25 KB, which is feasible even for users with slow

dial-up connections. With respect to program size, Stanford Folding is only about 371KB, and SETI@home is around 791KB. These applications run for a long time. The average computation time of each SETI@home job is about 6 hours.

### 6.3.2 Migration Strategies

There are two important issues for migration schemes: *when* to migrate the jobs and *where* to migrate the jobs. Traditional load sharing systems used central servers or high overhead information exchange to collect resource information about hosts in the system to determine when and where to migrate jobs [31, 88]. New scalable strategies are needed to guide migration decisions in a peer-to-peer system.

The optimal solution of *when* to migration the job requires accurate predication of future resource availability on all the hosts. Many researchers have addressed the CPU availability prediction problem for the Grid or for load sharing systems [10, 100, 28], but they all require a central location to collect and process the resource information. In our study, we assume there is no resource availability prediction and that migration is a simple best effort decision based primarily on local information, e.g. when the host becomes unavailable due to user activity.

The same resource availability issue exists for *where* to migrate the job. But the issue of *where* to migrate the job is also related to the scalable host discovery which we have discussed. The scheduler needs the host discovery to discover candidate hosts which are suitable to migrate the job to.

We compare several migration schemes that differ regarding when to migrate and where to migrate.

The options for when to migrate include:

- *Immediate migration.* Once the host is no longer available, the foreign jobs are immediately migrated to another available host.

- *Linger migration.* Linger migration allows foreign jobs to linger on the host for a random amount of time after the host becomes unavailable. After lingering, if the host becomes available again, the foreign job can continue execution on that host. Linger migration avoids unnecessary migration as the host might only be temporarily unavailable. Linger migration can also be used to avoid network congestion or contention for available hosts when a large number of jobs need to be migrated at the same time.

There are also two options for where to migrate the jobs:

- *Random.* The new host is selected in a random area in the overlay network. There is no targeted area for the search; the new host is a random host found in the area where the resource discovery scheme is launched.
- *Night-time machines.* The night-time machines are assumed to be idle for a large chunk of time. The Wave Scheduler uses the geographic CAN overlay to select a host in the night-time zone.

### 6.3.3 Peer-based scheduling strategies that utilize migration

The scheduling scheme has two distinct steps: *initial scheduling* and *later migration*. In initial scheduling, the initiator of the job uses host discovery to discover hosts satisfying the host selection criteria and schedules the job on the chosen host. The migration schemes also use host discovery to discover candidate hosts. Table 6.1 summarizes the difference between different basic migration schemes.

The *no-migration system* follows the SETI@home model. It uses the more relaxed host selection criteria: any *unclaimed* host can be a candidate.

**No-migration:** With no-migration, a client initially schedules the task on an unclaimed host, and the task never migrates during its lifetime. The task runs in screen-saver mode when the user is not using the machine, and sleeps when the machine is unavailable.



TABLE 6.1: Different migration strategies

Where to migrate	When to migrate	
	<i>Immediate Migration</i>	<i>Linger</i>
<i>Random Host</i>	Migration-immediate	Migration-linger
<i>Host in night-zone</i>	Wave-immediate	Wave-linger

WaveGrid and random-migrate all use migration schemes, which differs in where to migrate. WaveGrid migrates jobs to *available night-time* hosts, while random-migrate migrates jobs to *random available* hosts. In regarding to when to migrate, WaveGrid and random-migrate both adopt three different options: *immediate migration*, *linger migration* and *adaptive migration*.

**Immediate migration.** With immediate migration, the client initially schedules the task on an available host. When the host becomes unavailable, the foreign jobs are immediately migrated to another available host. In the best case, the task begins running immediately, migrates as soon as the current host is unavailable, and continues to run right away on a new available host.

**Linger migration.** With linger migration, a client initially schedules the task on an available host. When the host becomes unavailable, linger migration allows the task to linger on the host for a random amount of time. If the host is still unavailable after the lingering time is up, it then migrates. Linger migration avoids unnecessary migration as the host might be temporarily unavailable. Linger migration can also be used to avoid network congestion or contention for available hosts.

**Adaptive migration.** For initial scheduling, adaptive migration tries to find a host that is *available*. If it cannot, migration-adaptive schedules the task on an *unclaimed* host. When the host becomes unavailable, adaptive migration tries to migrate the task to a new host that is available. If it cannot find such a host, it allows the job to linger on the current host for a random amount of time and tries again later. A

cycle of attempted migration and lingering is repeated until the job finishes. Adaptive migration is designed to avoid rescheduling. However, it puts a bigger burden on the host since it may retry several times on behalf of the foreign task.

**Eager migration.** In this heterogeneous environment, we also designed an eager migration option, which can be added to all the basic migration strategies described above. With this aggressive migration model, after every *wakeup interval* a host does resource discovery to search for available hosts with better CPU power, while it is running the foreign jobs. If it can find such hosts, jobs will be migrated to these hosts immediately; otherwise the jobs continue to run on the current host, until it is time for them to leave the current hosts decided by the basic migration strategy.

## 6.4 Simulation-based Evaluation of WaveGrid

To evaluate the performance of WaveGrid, we compare it with a *no-migration system* and a *random-migration system*.

In our simulator, we implement the peer-based desktop grid systems using the following components: overlay network construction, host selection criteria, host discovery strategy, local scheduling policy, and scheduling scheme. The systems we evaluated use the same host discovery method and local scheduling policy, but differ in overlay network construction, host selection criteria and scheduling schemes. We use the scheduling and migration schemes described in section 6.3.3.

**Overlay network construction.** Both the *no-migration system* and the *random-migration system* use the CAN overlay network [78]. WaveGrid uses the CAN-based timezone-aware structured overlay network described above.

**Local Scheduling.** The local scheduling policy on a host determines the type of service a host gives to a foreign job that it has accepted. We use a *strict screensaver model*. We assume that one host can only accept one foreign job, and foreign jobs

can only run when it is admitted by local user policy such as there is no recent mouse/keyboard activity and the CPU utilization is low. When the host is available, the foreign job concurrently shares cycles with other local jobs.

**Host selection criteria.** A client uses its host selection criteria to select one host among multiple candidates. The primary selection criteria in a desktop grid system is host availability. As discussed in section 6.2, we use a strict host availability model. The following terms define the criteria regarding CPU availability which we use in the simulation. *Unclaimed* means that there is no foreign job on that host. *Available* means that there is no foreign job on that host and the host is idle. The host's local user policy described in local scheduling is used to decide whether the host is idle.

Different systems use different host availability criteria. The simple no-migration system relaxes this criteria to use any *unclaimed* hosts, while WaveGrid and random-migration try to schedule foreign jobs on *available* hosts for instant execution.

All three systems use CPU power as a criteria to choose the best host among all the candidates.

**Host Discovery.** The purpose of the host discovery scheme is to discover candidate hosts to accept the foreign job. WaveGrid does an expanding ring search in the targeted zone, while the other two systems do expanding ring search in the neighborhood of the client or a random area of the system. The benefit of the latter approach is to create a balanced load in case of a skewed client request distribution in the overlay.

## 6.5 Heterogeneous Host CPU Power Profile

We use a heterogeneous host CPU power profile derived from statistical data from BOINC project in this study [90]. BOINC is client-server-based Internet-wide cycle sharing system, which attracts millions of users. BOINC supports several scientific

applications. The most popular applications are searching for outer space intelligence(SETI@home), climate prediction(Climateprediction.net), studying about protein related diseases(Predictor@home), and searching for gravitational signals emitted by pulsars(Einstein@home). BOINC uses a credit-rewarding scheme to motivate hosts to donate cycles. Each time when a host returns a result, once the result is verified, the host is awarded some credits. Therefore the number of credits one hosts earns is directly proportional to the number of results it generates. For each application supported by BOINC, there is statistical data (<http://www.boincstats.com>) providing information such as total number of credits and average number of credits grouped by types of CPUs, or types of Operating Systems, or regions. There is also statistical data about individual user/host/team. Table 6.2 shows some sample entries from the BOINC statistic website, which is grouped by types of CPUs.

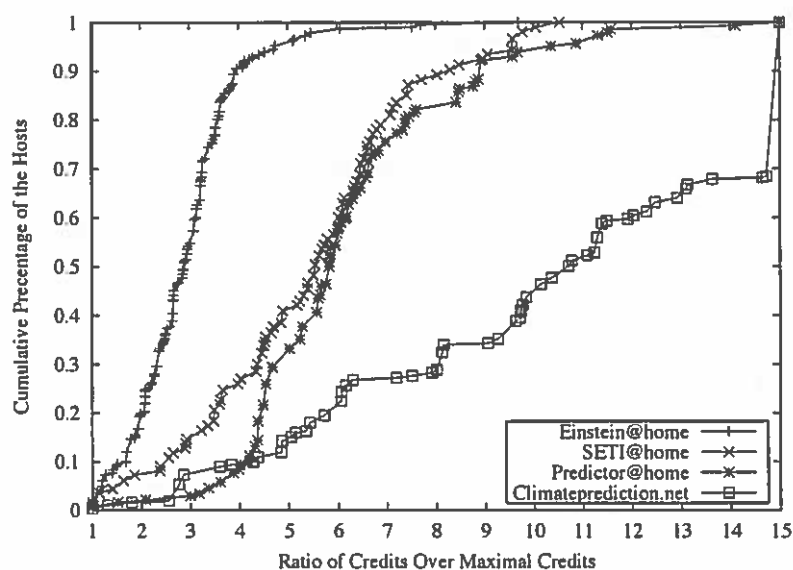
**TABLE 6.2:** Sample statistical data for SETI@home organized by types of CPUs, which groups hosts according to type of its CPU. (Note: "Average credit" is the average credit granted over the last few days to the hosts with this type of CPU.)

Pos.	CPU	#CPU	Total credit	Average credit	Credit per CPU	Average credit per CPU
1	Intel(R) Pentium(R) 4 CPU 3.00GHz	27,910	249,454,384.00	2,100,610.25	8,937.81	75.26
4	AMD Athlon(tm) Processor	13,934	47,707,040.00	356,019.28	3,423.79	25.55

We processed the statistical data about different CPUs in the system to generate the heterogeneous host CPU power profile via the following method. Hosts with same type of CPU are regarded as in the same group. We exclude those groups with a very small population, e.g. rare kind of CPUs such as high-end multiprocessors or vanishing types of CPUs. As the credits assigned to one host is directly related to the number of results returned by it, we use the average number of credits per CPU to predict the power of that type of CPU. For each CPU group, we compute the credit ratio as maximal average number of credits earned among all different groups over

average number of credits earned by hosts in that group, and the population of that group as a percentage of the total number of hosts. The smaller the credit ratio is, the better the CPU power of that group is.

Figure 6.3 shows the cumulative graph, in which each data point represents one group. The graph shows that for different types of applications, the distribution of CPU power is different, probably related to the structure of the particular program and the type of users attracted to that particular project. Except for Climateprediction.net, the other three applications exhibit a distribution similar to a normal distribution: a large number of groups have a similar ratio.



**FIGURE 6.3:** CPU powers of different CPU groups using empirical data from BOINC statistic (collected in Aug. 2005). The credits are averaged over all the CPUs in the same group.

We further convert the credit ratio into power rank. If the credit ratio of a group falls in the range of  $[k, k + 1)$ , the rank of that group equals  $k$ . The results in Figure 6.4 further confirm that the distribution of the CPU power is far from a uniform distribution. The majority of the hosts are clustered on one or a few ranks. We use the information in Figure 6.4 in our simulation study.

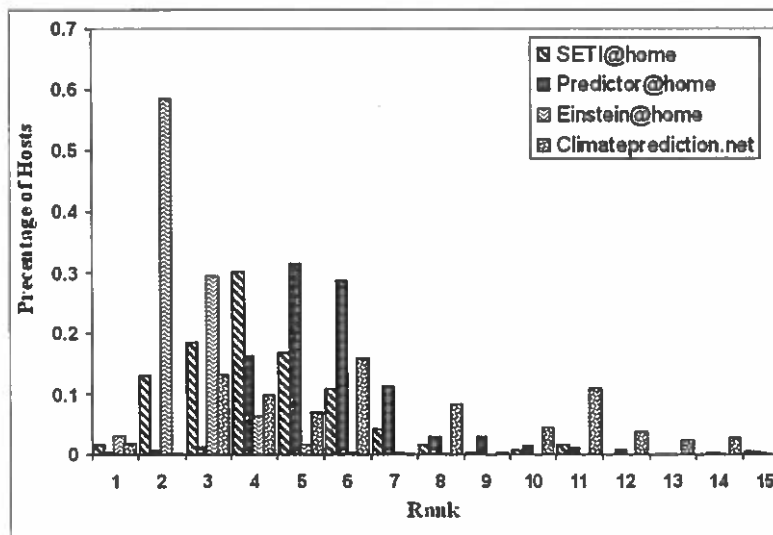


FIGURE 6.4: Percentage of different CPU groups with different ranks

The empirical data from BOINC also shows that the types of operating systems used by different hosts are quite uniform. About 89.1% of the hosts use windows operating system, and 7.6% of the hosts use linux operating system. In our simulation study, we assume that all the hosts use the same type of operating system or invoke the same type of virtual machines to run the foreign job. The BOINC data provides some information about the geographic distribution of the hosts, in which it shows that the majority of the hosts reside in north America. We believe this is biased information which derives from the fact that north America is the birthplace of the BOINC project, and thus it attracts many users in north America. According to statistic about the population of Internet users [73], 34.2% Internet users are in Asia, 28.5% Internet users are in Europe, and 23.4% Internet users are in North America. Therefore considering the fast growth of the Internet and the end host connection speed, the population of hosts in different parts of the world on a symmetric global-wise desktop grid system will be more equalized geographically than what is hinted by the BOINC statistic.

## 6.6 Simulation configuration

We use a 5000 node structured overlay in the simulation. Nodes join the overlay following the CAN protocol (or timezone-aware CAN protocol in the case of the Wave scheduler).

For simplicity, the Basic simulation configuration is used for in the preliminary study comparing different basic migration strategies. A more realistic simulation configuration is used when evaluating the performance of WaveGrid in a heterogeneous peer-based desktop grid environment.

### 6.6.1 Basic Simulation Configuration

**Profile of available cycles on hosts.** To evaluate the performance of different scheduling methods, a coarse-grain hourly synthetic profile is generated for each machine as follows: During the night-time (from 12pm to 6 am), the host is available with a very low CPU utilization level, from 0% to 10%. During the daytime, for each one hour slot it is randomly decided whether the machine is available or not. Finally, the local CPU load in a free daytime slot is generated from a uniform distribution ranging from 0% to 30%. We assume that when a host is running a foreign job, it can still initiate resource discovery for migration and relay messages for other hosts. The percentage of available time during the day varies from 5% to 95%. For simplicity, we assume all the hosts have the same computational power.

**Job workload.** During the simulation, a given peer can be both a client and a host. A random group of peers (10% to 90%) are chosen as clients. Each client submits a job to the system at a random point during the day. The job *runtime* is defined as the time needed for the job to run to completion on a dedicated machine. Job runtime is randomly distributed from 12 hours to 24 hours.

**Host discovery parameters.** We set the parameters of the resource discovery schemes to be scalable and to have low latency. The maximum number of scheduling attempts for random node label-based resource discovery is 5 times and the search scope for expanding ring search is 2 hops.

**Migration parameters.** The lingering time for the linger-migration models is randomly chosen in the range 1 hour to 3 hours. In the adaptive model, the linger time of a foreign job when it cannot find a better host to migrate to is also randomly chosen in the range 1 hour to 3 hours.

**Wave scheduler.** For the Wave scheduler, a 1x 24 CAN space is divided into 6 wavezones, each containing 4 time zones based on its second dimension. We studied the performance of a variety of strategies for selecting the initial wavezone and the wavezone to migrate to. The variations included (a) migrate the job to the wavezone whose earliest timezone just entered night-time, (b) migrate the job to a random night-time zone, and (c) migrate the job to a wavezone that currently contains the most night-time zones. The first option performs better than the others, since it provides the maximal length of night-time cycles. The simulation results presented in this paper use this option. However, it may be better to randomly select a nightzone to balance network traffic if many jobs simultaneously require wave scheduling.

## 6.6.2 Heterogeneous Environment and Rescheduling

The basic simulation configuration is modified and enhanced with the following option.

**Profile of available cycles on hosts.** When evaluating the performance of WaveGrid in a heterogeneous peer-based desktop grid system, the computation power of the hosts follows the heterogeneous model discussed in section 6.5. The amount of available time on each host is weighted by its CPU power. The normalized available time equals the unweighted available time times the CPU power divided by the median CPU power in the whole system.



**Job workload.** The *runtime* of a task is defined as the time needed for the task to finish on a dedicated machine with median CPU power rank over all the hosts in the system. It means that it will take a slow dedicated host more than the runtime to finish the job, and it will take a fast dedicated host less than the runtime to finish the job. The median CPU power is computed based on our empirical host CPU power profile, and the runtime is randomly distributed from 12 hours to 24 hours. Tasks belonging to the same application have same runtime.

**Rescheduling.** When a client fails to find an available host during initial scheduling or a host fails to find an available host to migrate the job to, it will try to reschedule the task after a random amount of time. The waiting time in the simulation is chosen between 1 to 2 hours, which includes the time to restart the task. The amount of time the rescheduling takes is included in the total execution time of the task.

**Migration parameters.** The migration delay is added into the total execution time. The migration delay includes time to discover available hosts, record the current status of the program, ship the code and data to the new host and restart the program on new hosts. For most of the simulation, the migration delay is 5 minutes, which is a fairly conservative figure for the type of applications suitable to run in WaveGrid.

## 6.7 Evaluating Different Migration Strategies

We conducted simulations to investigate the performance of the migration strategies described above and their effectiveness at reducing turnaround time, relative to a no-migration policy similar to SETI@home. We also evaluated the performance of the Wave Scheduler to see what gains are achievable through our strategy of exploiting knowledge of available idle blocks of time at night.

### 6.7.1 Simulation Metrics

Our evaluation of different scheduling strategies is focused on the turnaround time of a job, the time from when it first began execution to when it completes execution in the system.

In our study, a job is considered to have failed if the client fails to find a host satisfying host selection criteria, either when it initially tries to schedule the job, or when it later tries to migrate. Most of the performance metrics are measured only for those jobs that successfully complete execution. In this study, we do not model rescheduling, as we are interested in the success rate of the first scheduling attempt which includes the initial scheduling and the following successful migrations. The job completes in the shortest time if the client only needs to schedule the job once, so the slowdown factor measured this way show the peak performance of each migration scheme. Also it is interesting to see what percentage of jobs needs to be rescheduled under different migration models.

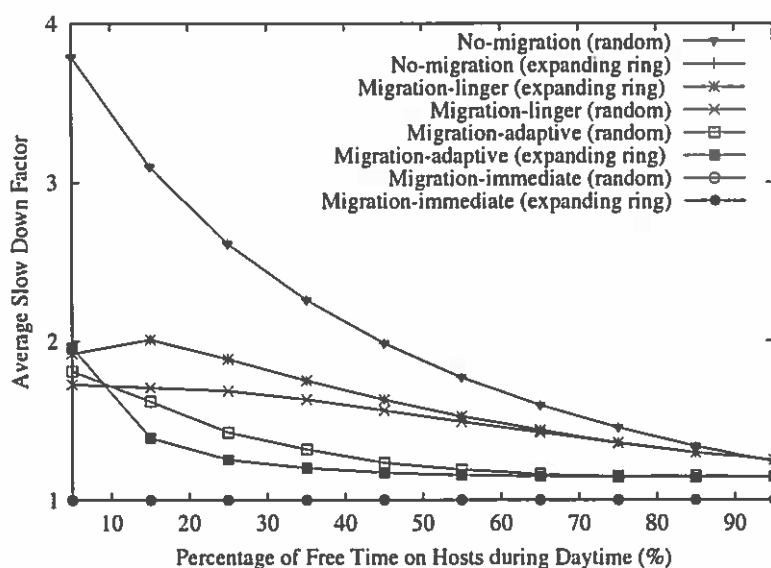


FIGURE 6.5: Average slowdown factor for no-migration vs. migration (The percentage of clients in the system is 20%)

The metrics used in the study are the followings:

- **% of jobs that fail to complete (job failure rate):** the number of failed jobs divided by the total number of jobs submitted to the system.
- **Average slowdown factor:** The slowdown of a job is its turnaround time (time to complete execution in the peer-to-peer cycle sharing system) divided by the job runtime (time to complete execution on a dedicated machine). We average the slowdown over all jobs that successfully complete execution.
- **Average number of migrations per job:** the number of times a job migrates during its lifetime in the system, averaged over all jobs that successfully complete execution.

We do not include migration and resource discovery overhead when plotting the average slowdown factor. The migration and resource discovery overhead do not make a visible difference in the results when migrations and resource discoveries are infrequent and the jobs run for a long time. We will analyze migration overhead, which dominates the computation overhead in the discussion of number of migrations. In the next simulation, we model the migration overhead including resource discovery latency, the time to transmit the data and code, the time to restore the program state and restart on the job on a new host.

## 6.7.2 Simulation Results

In this section, the legends in each graphs are ordered from top to bottom to match the relative position of the corresponding curves. Each data point is the average over 15 simulation runs.

## No-migration vs. migration

We first compare no-migration with the two basic migration schemes: migration-immediate and migration-linger. We measure the performance of these scheduling strategies as a function of percentage of free time on the hosts. When the percentage of free time on hosts increases on the x-axis, the load of the system decreases. We also examine the impact of the resource discovery scheme employed.

### *(a) The impact of migration on job turnaround times*

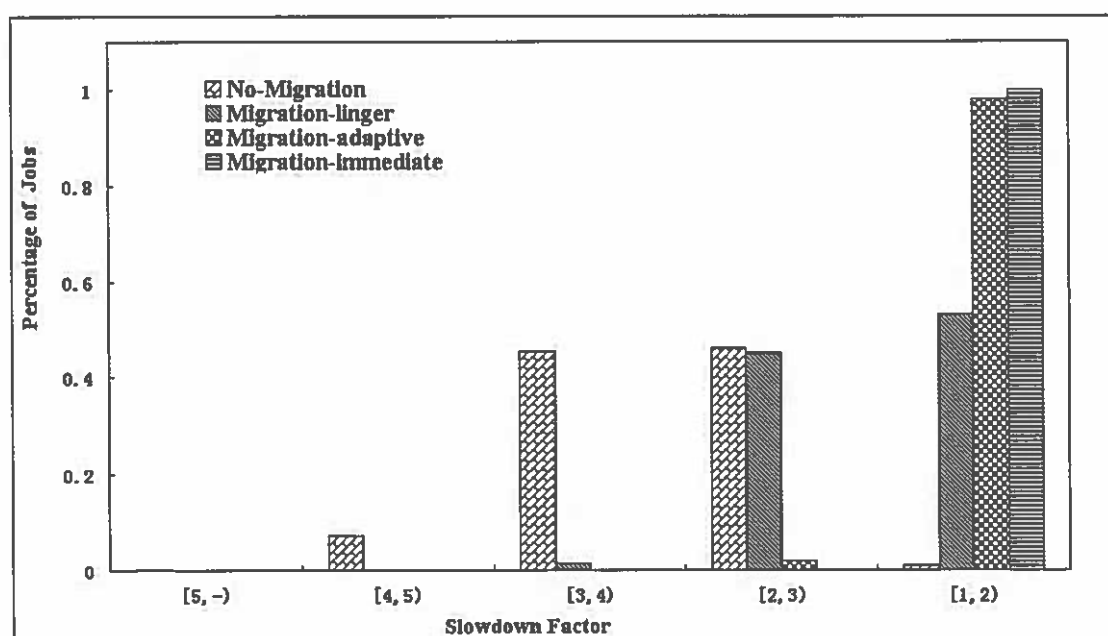
Figure 6.5 shows the average slowdown factor for successfully completed jobs as a function of free time on the hosts during the daytime hours. As expected, jobs progress faster with more available time on hosts during daytime. The performance of the no-migration strategy is clearly the worst since there is no effort made to avoid long waiting times when the host is not free. Note that the slowdown of migration-immediate (for both expanding ring and random host discovery) is always 1, since only jobs that successfully run to completion are considered. The success rate of different scheduling schemes will be discussed in section 6.7.2(b).

The performance of migration-linger is better than the no-migration strategy but worse than the others with its wasted lingering time associated with each migration. The performance of the adaptive models is the closest to the idealized migration-immediate schemes since it adaptively invokes the migration-immediate scheme whenever possible.

The slowdown factor is mainly influenced by the migration model used. However, for the linger and adaptive strategies, the resource discovery protocol also plays a role when the free time on the host is limited (e.g. when the percentage of hosts free time during daytime is less than 65%). We noticed that for the linger strategy, random performs better with respect to slowdown, but for the adaptive strategy, expanding ring performs better. This can be explained as follows: For comparable search times, expanding ring contacts more hosts than the random node label-based search, and therefore yields a higher successful migration rate. However, since with migration-

linger, every successful migration implies (wasted) lingering time, ultimately random has lower slowdown with its lower migration success rate. This observation is supported by Figure 6.17 which shows the average number of migrations for each strategy. For the adaptive strategy, expanding ring has lower slowdown than random as expected.

Figure 6.5 also shows that the slowdown factor for the no-migration strategy and for migration-immediate is insensitive to the resource discovery schemes.



**FIGURE 6.6:** histogram of slowdown factors of successfully finished jobs (The percentage of clients is 20% and the percentage of free time on hosts is 15%)

Figure 6.6 further confirms the improvement of turnaround time when using a migration model under heavy load. The majority of jobs scheduled by no-migration scheduling experienced a slowdown greater than 2 and in the extreme case, jobs may experience slowdown greater than 5. The majority of jobs scheduled by migration-adaptive and migration-immediate have small slowdown or no slowdown at all.

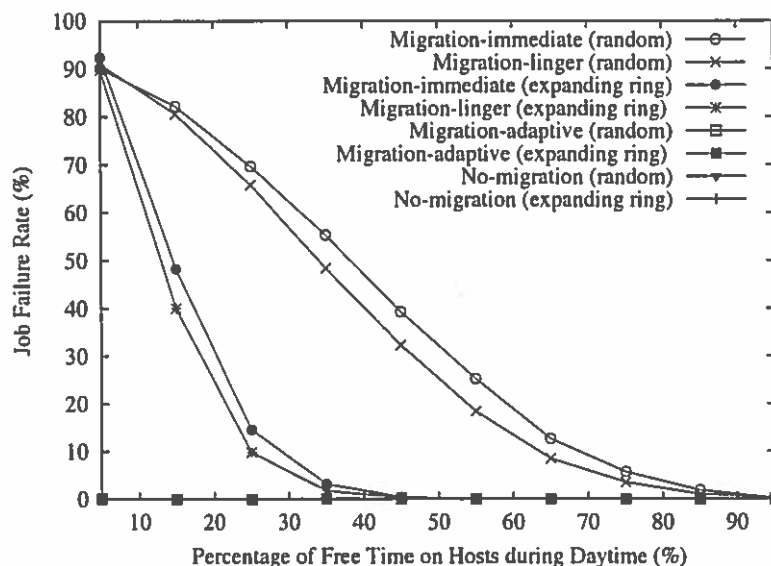


FIGURE 6.7: % of jobs that fail to complete (The percentage of clients in the system is 20%)

(b) *The impact of migration on successful job completion*

The above results regarding slowdown factor cannot be considered in isolation. In particular, it is necessary to also consider the job failure rates, i.e. percentage of jobs for which a host satisfying host selection criteria cannot be found either in initial scheduling or in migration.

Figure 6.7 shows the percentage of jobs that failed to finish assuming the same simulation configuration as in Figure 6.5. For the two strict migration models (migration-immediate and migration-linger) which only use local availability information, the failure rate is extremely high. The adaptive models, which use a small amount of global information at migration time, have dramatically fewer failed jobs – close to zero.

There is a tradeoff between the job turnaround time and percentage of jobs that successfully complete. The strict models have the lowest turnaround time, but high failure rates when free time on the hosts is limited. The adaptive model performs best because it achieves low turnaround time with highest job completion rate.

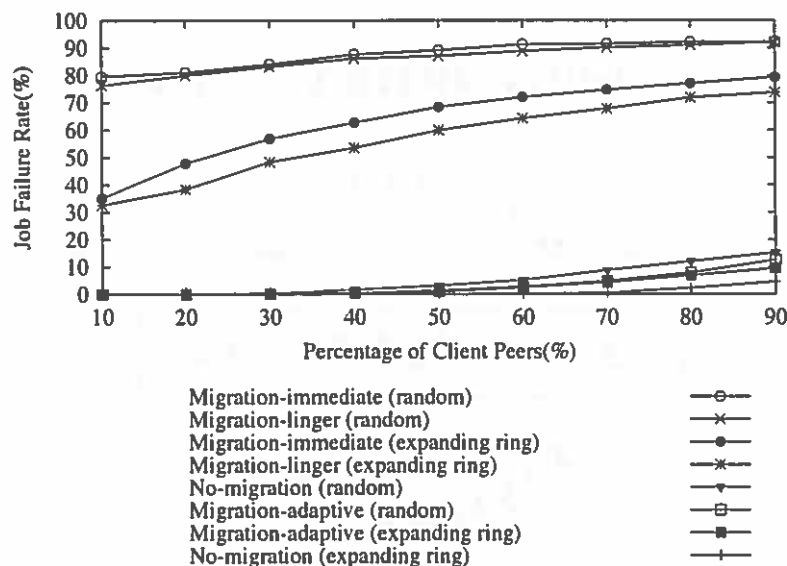


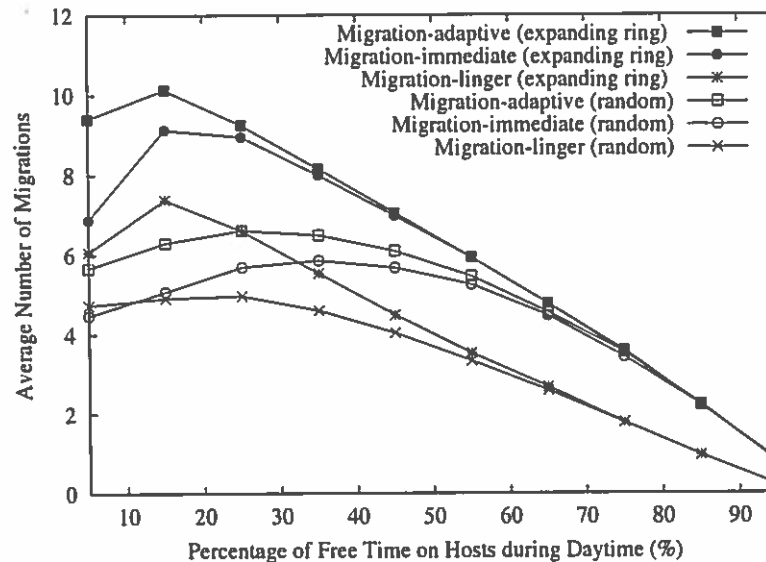
FIGURE 6.8: % of jobs that fail to complete (The percentage of free time on the hosts during the day is 15%)

When the number of client requests increase, there will be intense competition for free hosts. When the free time on these hosts is small, the situation is even worse. Figure 6.8 shows that the failure rate of all scheduling strategies increases with the increasing number of client requests. The persistently high failure rate of migration-immediate makes it impractical for real applications when the available resources are very limited.

The simulation results show that with abundant host free time, the failure rate of migration-adaptive using an expanding ring search is even slightly lower than the no-migration scheme.

### (c) Number of migrations during job lifetime

Figure 6.17 shows that the average number of migrations varies with the different migration scheduling strategies. The graph shows that when the percentage of host free time increases, the number of migrations increases first and then decreases. The reason is that there are two factors that influence the number of migrations: the



**FIGURE 6.9:** Average number of migrations for successfully finished jobs (The percentage of clients in the system is 20%)

number of currently available hosts and the length of free time slots on the hosts. With more available hosts, there is higher chance of migration success and therefore a larger number of migrations. With longer free time slots, the need for the jobs to migrate is reduced. With higher percentage of free time, the amount of currently available hosts increases and the length of free time slots also increases.

We can demonstrate that migration overhead is low using the same graph. In early morning or late night, the network traffic in the Internet is usually light. Therefore the network connection from the end-host to the Internet is usually the bottleneck link when downloading or uploading data. In the following computation, we assume the upload bandwidth of the host is 256kb, which is the bandwidth of slow-end DSL users and download bandwidth is higher than upload bandwidth with DSL. If the amount of data to be transmitted during the migration is 1MB, the slowdown factor of migration schemes will increase by at most 0.005 when the running time of the job is longer than 12 hours. Even when the amount of data to transmit is 20MB, which is quite large for a scientific computation, the influence is at most 0.1. The time



overhead of resource discoveries is much smaller and negligible compared with that of migration. In next section, we present simulation results with migration overhead using a heterogenous host CPU profile.

### Performance of Wave Scheduler

This section presents the evaluation of the Wave Scheduler. In order to focus on the difference between migration strategies, we only describe results with the resource discovery method set as expanding ring search. (Simulations with random node label-based discovery show the same relative performance.)

#### (a) Impact of Wave scheduler on turnaround time

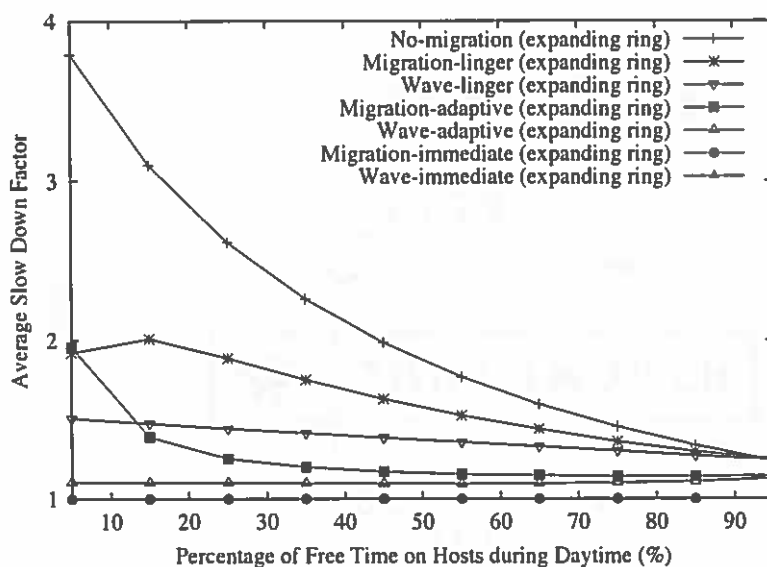


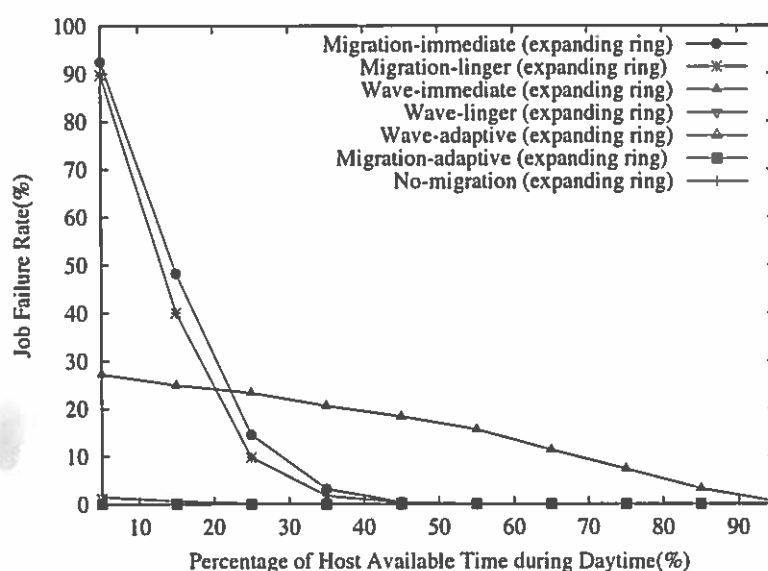
FIGURE 6.10: Wave Scheduler: Average slow down factor (The percentage of client request is 20%)

Figure 6.10 shows that the turnaround time of jobs with Wave Schedulers is lower than other migration schedulers. Jobs progress faster with the Wave Scheduler because it can identify hosts with potentially large chunks of available times. The turnaround time of wave-adaptive is consistently low, while the turnaround time

of migration-adaptive is significantly higher when the amount of free time is small. When the percentage of free time on hosts is 15%, the turnaround time of jobs under wave-adaptive is about 75% of that under migration-adaptive.

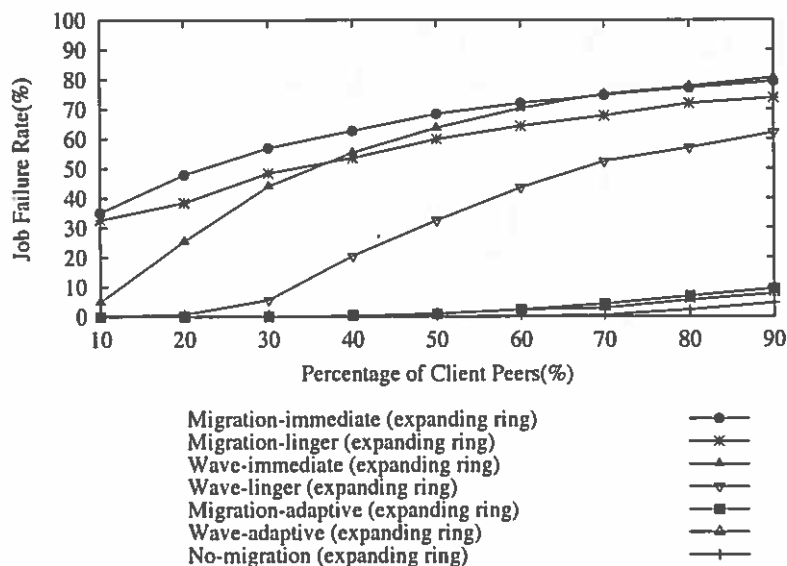
*(b) Impact of Wave scheduler on successful job completion*

The percentage of jobs that fails to complete using the Wave scheduler is influenced by two factors. Wave identifies available hosts with large chunks of future free time. However, if the ratio of requests to the number of such hosts is limited, there will be scheduling conflicts.



**FIGURE 6.11:** % of job that fail to complete (The percentage of clients in the system is 20%)

When the free time on hosts is limited, the Wave scheduler does better than other migration schemes since it was designed for this exact situation. (see Figure 6.12). The intensive contention for night-time hosts is relieved by wave-adaptive, which adapts to the amount of free time by continuing to stay on the hosts in case of contention. The job failure rate of wave-adaptive is competitive with the no-migration model and slightly lower than with migration-adaptive.



**FIGURE 6.12:** % of job that fail to complete (The percentage of available time on the hosts during the day is 15%)

Figure 6.11 shows the percentage of jobs that failed to finish under the same simulation configuration as in Figure 6.10. The job failure rate of the Wave scheduler is relatively higher than others when the percentage of free time on hosts increases, as wave-immediate uses strict rules about using night-time hosts and this cause contention. The other two wave scheduler strategies perform as well as migration-adaptive and the no-migration strategy.

*(c) Number of migrations during job lifetime with Wave scheduler*

Figure 6.13 compares the average number of migrations of successfully finished jobs with the wave migration strategies versus the standard migration. As we expected, jobs scheduled with the Wave scheduler finished with fewer migrations, because it exploits the long available intervals at night while the others may end up using short, dispersed time slots during the day. As in the discussion about migration overhead in section 6.7.2(c), the migration overhead of Wave Scheduler is even smaller compared with the standard migration schemes and therefore it is acceptable for jobs with long running time.

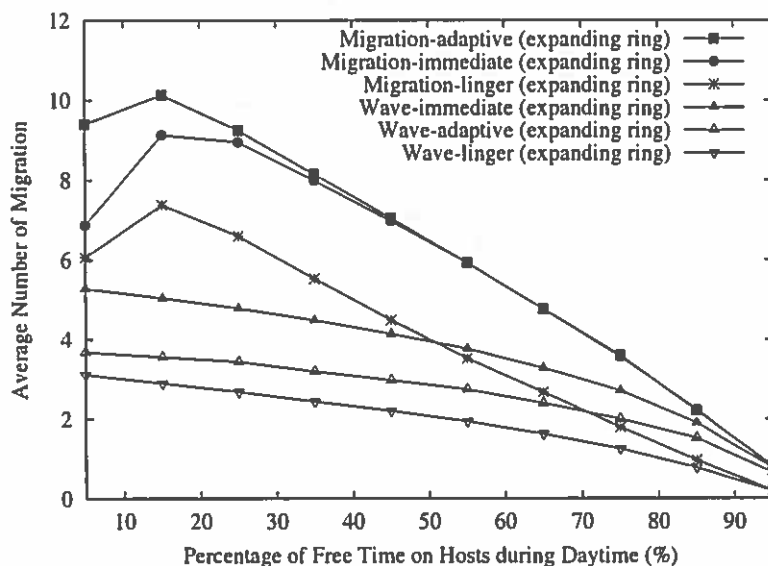


FIGURE 6.13: Average number of migrations (The percentage of client in the system is 20%)

## 6.8 Simulation Experiments using a Heterogeneous CPU profile

In this section, we present simulation results evaluating the performance of WaveGrid. We compare WaveGrid with two other peer-based desktop grid systems, *no-migration* and *random-migration*, using varied migration schemes described in section 6.4 under an empirical heterogeneous host CPU power model. We tested four metrics to evaluate the performance of the system: slowdown, makespan, number of rescheduling attempts, and migration overhead.

### 6.8.1 Simulation Metrics

We use the following metrics for a comprehensive study of WaveGrid.

- **Average slowdown factor:** The slowdown of a task is its turnaround time (time to complete execution in the peer-to-peer desktop grid system) divided by

the task runtime. Turnaround time includes both migration time and waiting time due to rescheduling. We average the slowdown over all tasks. The slowdown of one task can be less than 1 in this scenario as the runtime of the task is defined using the median CPU power of the system while the actual execution time on one host is weighted by its CPU power.

- **Average makespan:** The makespan of an application is the time the first task belonging to that application is submitted until the last task finishes divided by runtime of the task (Tasks in one application have equal runtime.). We then average the makespan over all applications.
- **Average number of migrations per task:** the number of times a task migrates during its lifetime in the system, averaged over all tasks.
- **Average number of retries per task:** the number of times a task is rescheduled during its lifetime in the system, averaged over all tasks.

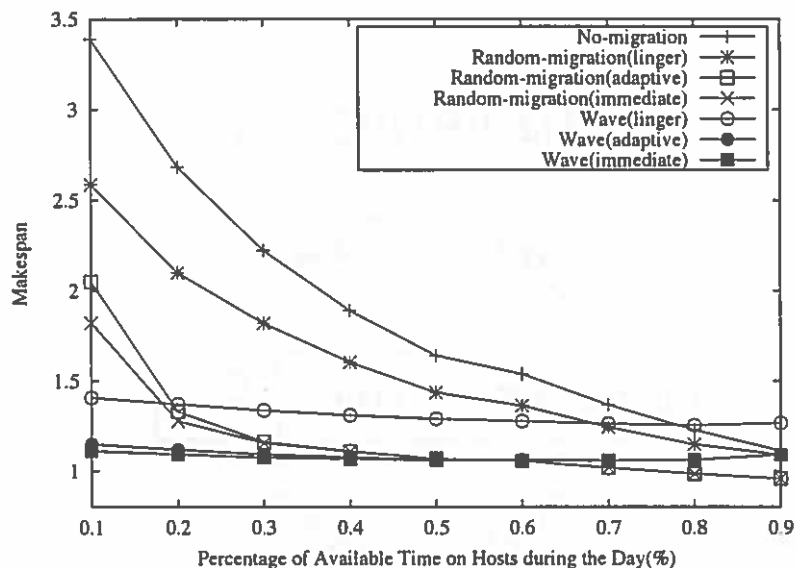
## 6.8.2 Simulation Results

Our simulation study investigates the performance gains achievable through timezone-aware organization of the hosts, efficient migration and scheduling strategies that consider the heterogeneity of the system.

In this section, the legends in each graphs are ordered from top to bottom to match the relative position of the corresponding curves. Each data point is the average over 15 simulation runs.

### 1) Overall Performance of WaveGrid

The overall performance of WaveGrid is stable and is better than the other systems with limited available cycles, as WaveGrid efficiently organizes hosts according to its timezone and migrates using this timezone information.



**FIGURE 6.14:** Average makespan vs host availability (The percentage of clients is 20%)

Figure 6.14 shows the makespan of applications when the percentage of available time on hosts varies. Figure 6.15 shows the slowdown of applications in the same scenario. All systems perform better as the amount of available time increases. No-migration performs the worst, as it does not try to find available hosts for continuing execution of the foreign applications and a lot of time is wasted in waiting for the hosts to be available again.

The performance of WaveGrid is quite stable over different host availabilities. When the available time on hosts during the day changes from 90% to 10%, the performance of WaveGrid degrades less than 10%. In contrast, the performance of random-migrate degrades more than 85%.

WaveGrid performs better than all the other systems as it makes efficient use of large chunks of night-time cycles, when the available cycles are limited. Random-migration improves with increasing host availability. When the hosts are mostly available during the day, it performs slightly better than WaveGrid. The reason for

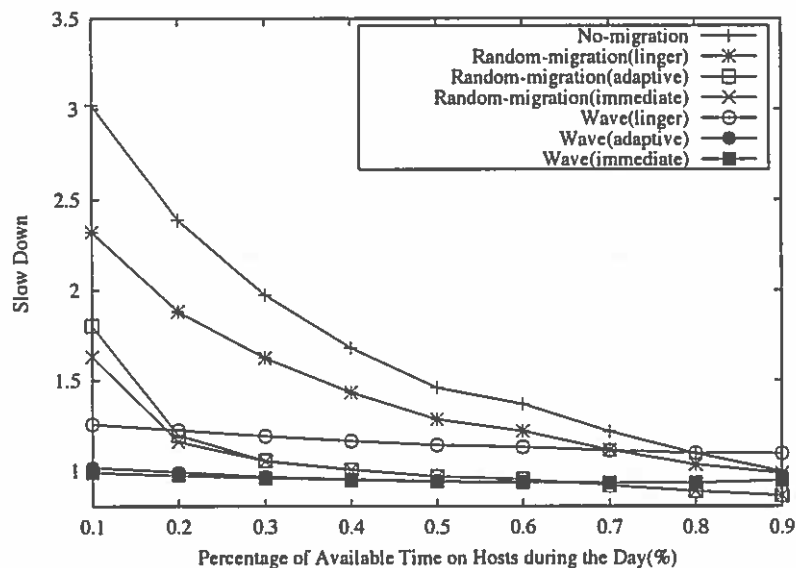
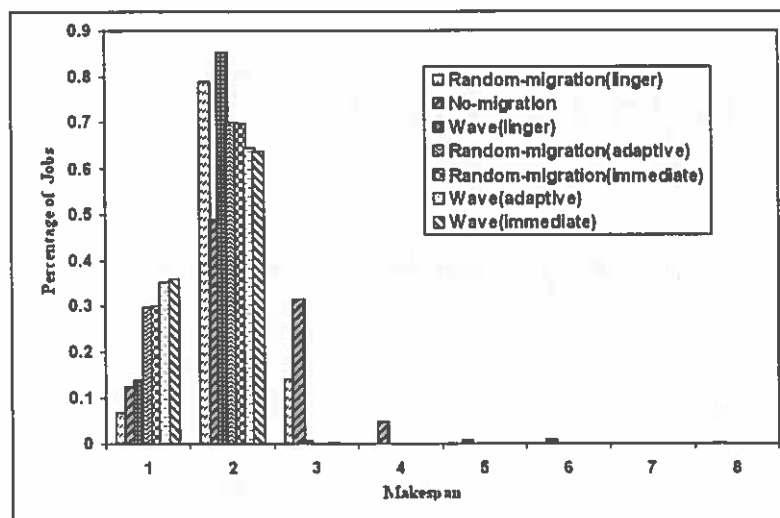


FIGURE 6.15: Average slowdown vs host availability (Percentage of clients is 20%)

that is random-migration selects candidate hosts from a larger pool than WaveGrid, i.e. it is not restricted to the hosts in the next wavezone. Thus it has higher chance to identify some host with better CPU power in this heterogeneous system.

Figure 6.16 confirms that systems utilizing migration perform better than the no-migration option. The makespan of applications in the no-migration system has a long-tail distribution, and in the extreme case the makespan of one application is as high as 8. In contrast, the makespan of applications in migration-based systems, except for those using linger migration strategies, is less than 2. WaveGrid using immediate migration strategy performs the best with the highest percentage of applications having a makespan which is less than 1. Recall that makespan can be less than 1, since it is defined in terms of a host with median CPU power.



**FIGURE 6.16:** Histogram of distribution of makespan (Percentage of clients is 20%. Percentage of available time on hosts during the day is 40%.)

## 2) Effective migration in WaveGrid

The migration schemes used by WaveGrid effectively reduce the number of migrations, migration delay and number of rescheduling attempts.

Figure 6.17 shows the average number of migrations. As expected, jobs scheduled with WaveGrid finished with fewer migrations, because it exploits the long available intervals at night while the others may end up using short, dispersed time slots during the day. WaveGrid minimizes disturbance to hosts as it uses fewer hosts in migration and therefore contacts fewer hosts for resource discovery, which is important to cycle donors.

The graph also shows that when the percentage of host availability increases, the number of migrations in random-migration increases first then decreases. The reason is that there are two factors that influence the number of migrations: the number of currently available hosts and the length of free time slots on the hosts. With more available hosts, there is higher chance of migration success and therefore a larger



number of migrations. With longer free time slots, the need for the jobs to migrate is reduced. With higher percentage of free time, the amount of currently available hosts increases and the length of free time slots also increases.

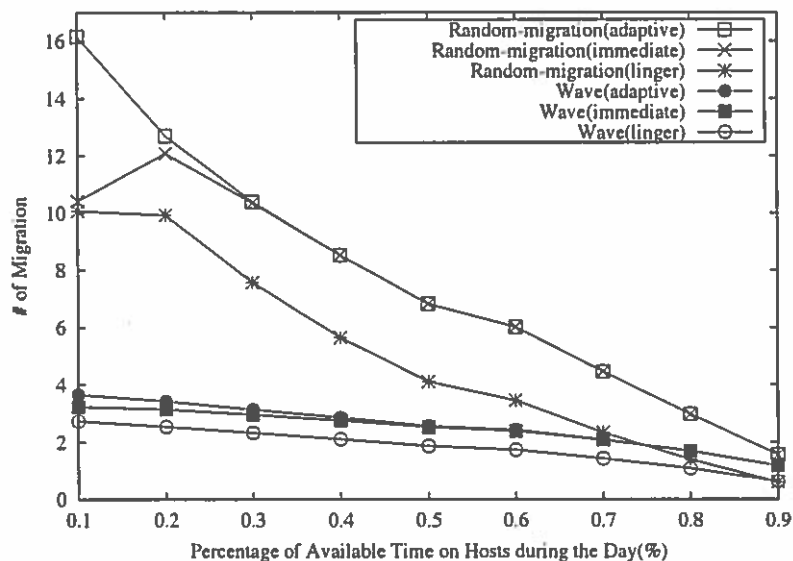


FIGURE 6.17: Average number of migrations (Percentage of clients is 20%)

Figure 6.18 shows how makespan varies as a function of migration delay. It shows that makespan increases with the increasing migration delay. The migration delay has a much larger impact on the performance of random-migration than WaveGrid. When the migration delay increases from 2 minutes to 22 minutes, the makespan in random-migration(immediate) increases about 19.4%, while the makespan in WaveGrid(immediate) only increases about 6%.

Figure 6.19 shows that WaveGrid has much fewer scheduling retries than random-migration. The reason for that is in random-migration the scheduler has no knowledge about where to find potential hosts with available cycles.

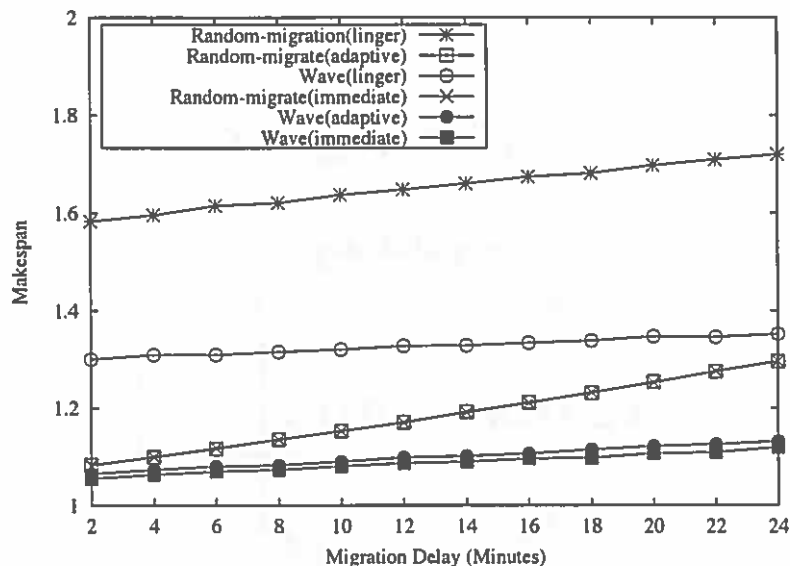
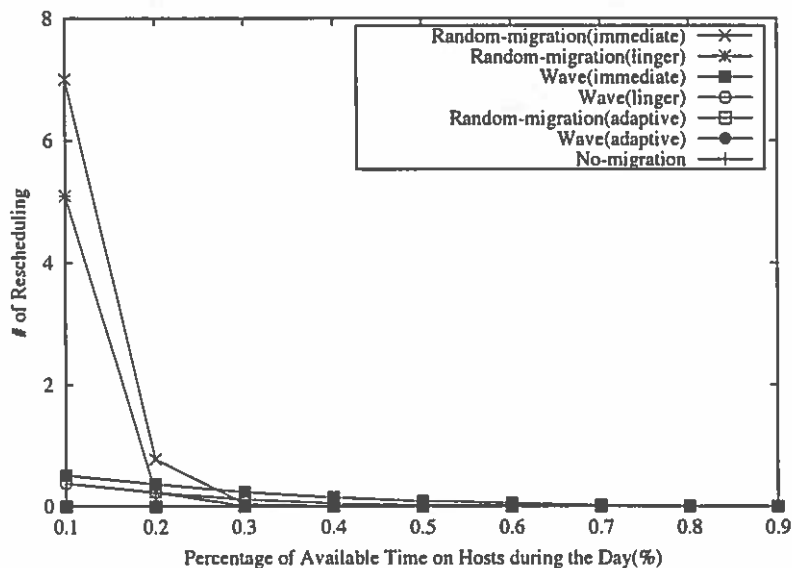


FIGURE 6.18: Average makespan when the migration delay varies (Percentage of clients in the system is 20%. Percentage of free time on hosts during the day is 50%.)

### 3) Scheduling in a Heterogeneous Environment

Performance of all systems improve when the scheduling strategy chooses the most powerful host from multiple candidates. Performance of WaveGrid and random-migration can be further improved when the eager migration strategy is used.

Figure 6.20 shows the slowdown of applications when the scheduler selects one random host instead of the most powerful host where there are multiple candidates (Only selected strategies are shown due to limited space). Figure 6.21 shows the percentage of performance improvement for all the strategies. Our results show that WaveGrid's superior performance is not greatly impacted by using the host selection strategy based on CPU power, although it does improve performance. We also see that this host selection strategy yields significant improvements for no-migration and random-migration.



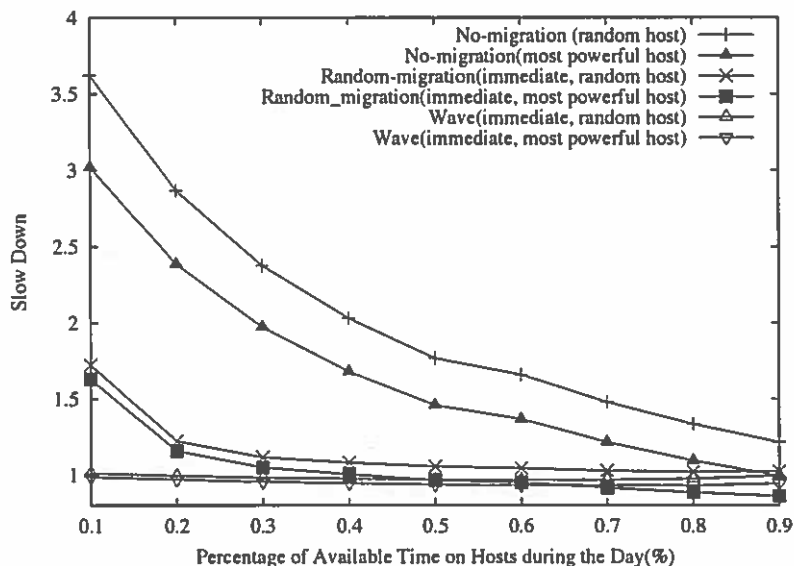
**FIGURE 6.19:** Average number of retries during the job execution (Percentage of clients is 20%)

Figure 6.22 shows that when turning on the option of eager migration, the performance of both WaveGrid and random-migration improve. Figure 6.23 shows the percentage of performance improvement. It shows that eager migration improves performance of both systems, while eager migration has a bigger impact on random-migration as there is more room for improvement.

## 6.9 Extensions to WaveGrid

A number of extensions to WaveGrid can enhance its performance regarding scheduling efficiency and fault-tolerance.

The *night-time* concept can be extended to any long interval of available time. The overlay does not necessary need to be timezone based but can be organized based on available time intervals. For example, a user in Pacific Time timezone can register



**FIGURE 6.20:** Average slowdown vs host availability when using random host selection (Percentage of clients is 20%)

her home machine in a wavezone containing hosts idle from 0:00-10:00 GMT, when she knows she is at work during the daytime. Wave scheduler can even accept more complicated user profiles, which indicates idle time slots.

WaveGrid can also be easily leveraged to handle greater heterogeneity of the hosts in the system. Information such as operating system, memory and machine types can be represented by further dividing the node label space or adding separate dimensions. Checkpointing techniques can be added to WaveGrid to provide fault-tolerance, using the CAN file sharing protocol to organize the hosts into a distributed file system. The host periodically stores the checkpointing information in the underlying CAN DHT, and the initiator of the application can retrieve the information if the host fails. The key to the data can be the address and port number of the initiator and the name of the application.

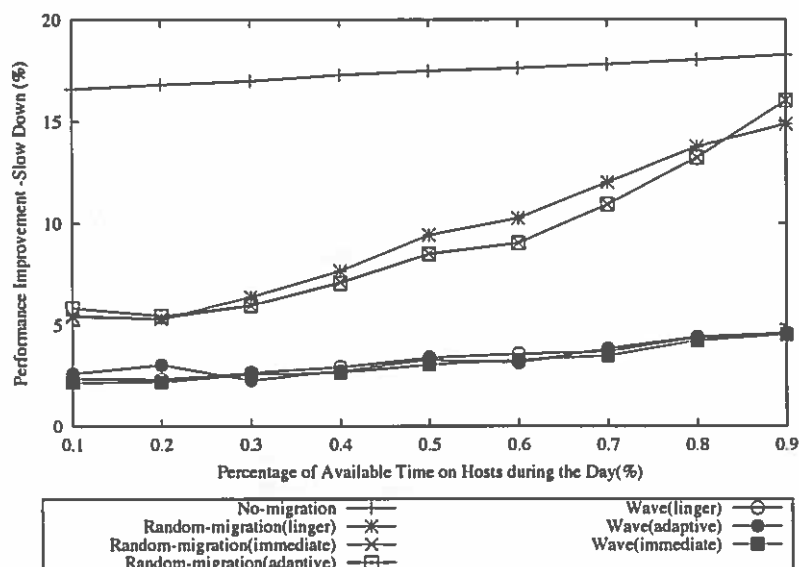
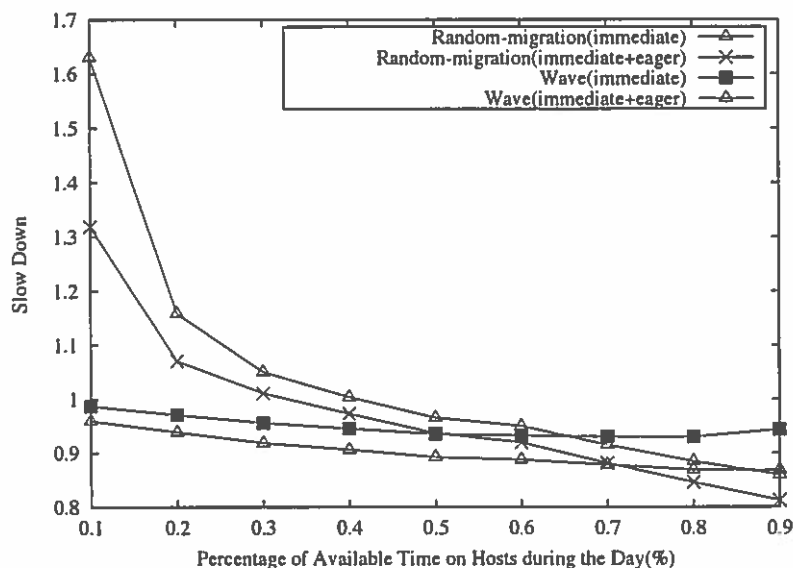


FIGURE 6.21: Percentage of performance improvement when using the most powerful host selection instead of a random host selection (Percentage of clients is 20%)

When implementing WaveGrid, it is possible that the geographic population of hosts on the Internet is unbalanced. Thus, we may need to merge several timezones with small population into one Wavezone or even to skip timezones which are mostly in the ocean, such as the timezone where Fiji is. The hosts in these timezone will be instructed to join another timezone by the WaveGrid overlay construction protocol.

## 6.10 Conclusion

We propose a novel heterogeneous scalable fast-turnaround desktop grid system, WaveGrid. WaveGrid allows hosts to register themselves in a structured overlay network according to their long idle time slots at night. A client can quickly target an area to search for available hosts without sending a large amount of messages. Application schedulers migrate jobs from busy hosts to idle hosts with potentially large



**FIGURE 6.22:** Average slowdown when using the eager migration strategy (Percentage of clients is 20%)

chunk of available time, maximizing utilization of available cycles. The simulation results show that WaveGrid outperforms other systems with respect to turnaround, stability and minimal impacts on hosts.

Heterogeneity is inherent to the nature of peer-based desktop grid systems. To accommodate heterogeneity of the system, we used a host selection criteria based on CPU power. We further designed an eager migration scheme which actively seeks for powerful hosts to migrate the jobs to. In our simulation, we used a heterogeneous host CPU power model based on empirical data from BOINC. This model is helpful for designing and evaluating other global desktop grid systems.

WaveGrid integrates several layers of components centered around the goal of fast turnaround: A timezone-aware overlay network which organizes hosts for fast resource discovery, and scheduling and migration using heterogeneous host information. We believe the design approach of WaveGrid is promising and can be used for other large-scale peer-based desktop grid systems.

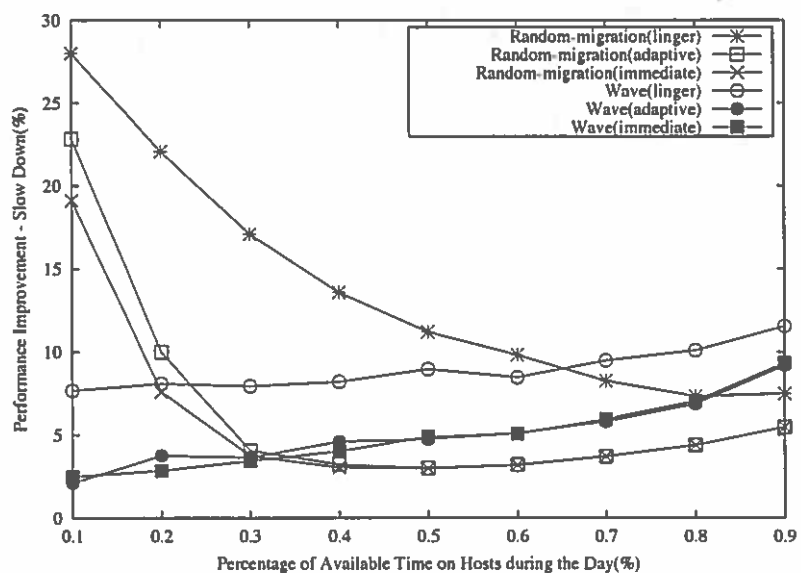


FIGURE 6.23: Percentage of performance improvement when using eager migration (Percentage of clients is 20%)

## CHAPTER 7

# Conclusion and Future Work

### 7.1 Contribution

A scalable, light-weight, and easy to access peer-based desktop grid system which harnesses idle cycles donated by hosts at the edge of the Internet can serve the increasing need for computational cycles required by a wide range of applications. These range from scientific applications to game and multimedia processing. In contrast to previous cycle sharing systems, peer-based desktop grid systems do not use central servers, and can thus achieve much larger scale. My dissertation research focuses on the challenge of fast turnaround scheduling in a global desktop grid system through two key techniques: fast resource discovery and migration.

We have designed a general peer-based desktop grid system [67], which is among the first peer-based cycle sharing architectures proposed in the literature. Our architecture adapts techniques from the field of peer-to-peer networks to cycle sharing systems to remove the bottleneck, cost, and vulnerability of the previous central-server based approaches. This dissertation defines a generic peer-based cycle sharing architecture and addresses the challenge of scheduling CPU intensive workpile jobs in this type of system.



Scheduling for fast turnaround in peer-based desktop grid systems is an important and open research field. Many applications have fast-turnaround requirements, such as weather forecasting, research for a cure of a disease, and scientific applications with research reporting deadlines. However, current Internet-wide cycle sharing production systems can only provide best-effort scheduling which can yield poor turnaround time. To date, minimal efforts have been made to improve scheduling methods for better turnaround time in peer-based cycle sharing systems.

We note that to solve the problem of scheduling for fast turnaround, we need to find the most scalable and efficient resource discovery method for locating idle hosts in the Internet. Most research about resource discovery in peer-to-peer networks is focused on content location in file sharing systems. We are the first to conduct a comprehensive simulation study of generic resource discovery methods (including expanding ring search, random walk, advertisement-based search and rendezvous point based search) for locating idle cycles in peer-based desktop grid systems [103]. Our results show that the efficiency of a rendezvous point scheme is better than the other schemes and is comparable to the central server scheme with a job completion rate within 95% of the job completion rate of the central server scheme. In addition, the rendezvous point scheme has much lower message overhead than the other schemes. Therefore, rendezvous point based resource discovery performs best for locating available idle hosts and for scheduling performance, considering both efficiency and overhead.

To further investigate the design of rendezvous point based schemes, we designed a dynamic rendezvous point selection scheme for structured overlay networks using a regular node label scheme. Although many rendezvous point selection schemes have been proposed for unstructured overlay networks [56, 46], these schemes cannot guarantee an even distribution of rendezvous points so that all non-rendezvous points have low access to the rendezvous points. Those schemes rely on expanding ring search to find rendezvous points, and thus incur high search overhead; another disadvantage of these algorithms is that each client only knows a few rendezvous points in the

system. Our rendezvous point scheme, SORPS, can guarantee a stable ratio of rendezvous points to non-rendezvous points and low access from non-rendezvous points to rendezvous points [68]. Fault-tolerance can be achieved via redundancy; combining node qualifications into the selection scheme can further improve the performance of SORPS.

Although SORPS can achieve even distribution of rendezvous points and low access from non-rendezvous points to rendezvous points, there are still some inherent drawbacks to a physical rendezvous point scheme. These include the burden for peers to function as rendezvous points, the potential of security attacks, and complicated fault-tolerance schemes. We next developed a unique new approach, *virtual rendezvous points*, and the notation of a *resource aware overlay network (ROAN)*. In a ROAN, the node labeling space is divided into zones according to the resource information and hosts join the right zone according to their resources. Clients' request can be automatically routed to matching hosts using bounded latency DHT routing associated with the underlying structured overlay. The virtual rendezvous point scheme has the merits of efficiency and low message overhead, and further eliminates the vulnerability to failure and potential security problems associated the physical rendezvous point scheme.

The culmination of our work and the most important contribution in this dissertation is WaveGrid, a scalable heterogeneous scheduling architecture for fast turnaround in peer-based desktop grid systems. The key components of WaveGrid are (1) a timezone-aware structured overlay network (a specialized ROAN) for fast resource discovery and (2) migration schemes for access to continuing available cycles in the Grid. We have investigate a range of different migration schemes with options for where to migrate and when to migrate. For when to migrate, we studied both immediate migration and migration after lingering on the current host for a while. For where to migrate, we studied migration to a night-time zone and migration to random hosts. The results show that immediate migration to night-time zone hosts performs

best when considering both slowdown and number of migrations. We then use a heterogeneous host profile based on statistical data derived from a production system to perform a comprehensive evaluation of WaveGrid. Our simulation results show that WaveGrid performs better than random migration and no migration with respect to slowdown and low migration overhead. In addition, the performance of WaveGrid is quite stable over different host availabilities: when the available time on hosts during the day changes from 90% to 10%, the performance of WaveGrid degrades less than 10%. In contrast, the performance of random migration degrades more than 85%. Moreover, the migration overhead of WaveGrid is much lower than that of random migration. The simulation results also show that when considering heterogeneity of the system, the performance of the scheduler is much improved.

We have reported a comprehensive body of research on scheduling for fast turnaround in peer-based desktop grid systems in this dissertation. Our unique approach seeks to achieve the performance of a dedicated system in a scalable autonomous peer-to-peer cycle sharing architecture. Our system combines the features of an easy-to-assemble and light-weight peer-to-peer architecture with advanced resource discovery and scheduling schemes for fast turnaround for long running CPU intensive application. In this dissertation, we have made the following contributions:

- **A comprehensive study of generic resource discovery methods in peer-based cycle sharing systems.** Our simulation results show that rendezvous point performs the best considering both efficiency and search overhead.
- **A rendezvous point selection method in structured overlay networks.** We designed SORPS, a rendezvous point selection method in structured overlay networks, which can achieve a stable ratio of rendezvous points to non-rendezvous points and even distribution of rendezvous points.

- **Virtual rendezvous points and resource-aware overlay networks.** We have developed the notion of a resource-aware overlay network (ROAN) using virtual rendezvous points based on structured overlay networks. Fast and efficient resource discovery can be achieved using a ROAN to embed resource information into the topology of the overlay network.
- **Fast turnaround scheduling using a timezone-aware ROAN and migration.** Our unique approach of using migration in WaveGrid coupled with timezone-aware resource discovery results in superior performance with respect to fast turnaround time.

An open and light-weight peer-based desktop grid system can potentially attract a large number of idle hosts on the Internet. Our research directly benefits a wide range of applications which need a huge amount of cycles and have fast turnaround requirements. Results will be produced much faster in WaveGrid than in the previous Internet-wide cycle sharing systems. We believe that the design of future peer-based desktop grid systems can greatly benefit by the results in this dissertation; especially by using the ROAN and migration to achieve better scheduling performance.

## 7.2 Future Work

### 7.2.1 Improvement to WaveGrid

A number of enhancements can further improve the performance of WaveGrid for fast turnaround. As we have described in Chapter 6, the night-time concept in WaveGrid could be extended to any long chunk of idle cycles, such as day-time slots on a home machine. More resource information such as memory size and type of operating system, can also be built in the ROAN of WaveGrid.

In addition to these direct extensions of WaveGrid, we also propose research on fault-tolerance and fairness of the system in order to further improve the turnaround time. The goal is to waste minimal amount of computational work when faced with hosts' failures or departure, and to prevent selfish clients from trying to grab all the cycles, resulting in worse performance for the other clients.

### 7.2.2 Peer-to-peer Checkpointing

Fault-tolerant scheduling is a big challenge in peer-to-peer cycle sharing systems in which a peer fails or leaves. The foreign job running on that host stalls and the previous computational work may be lost, thus delaying completion of the job. A client may restart the task when it detects the host failure via loss of heartbeat messages. This is the simplest method but it wastes cycles and causes long delays for the task. We propose to explore methods such as replication and checkpointing.

With replication, the client sends out several duplicate copies of the same task in the hope that one or some of them will complete successfully. Replication does not save total cycles but it provides faster turnaround than restarting the task, and it works naturally with result verification methods that use replication and cross-check. An important factor in replication is how many duplicated copies the client needs to send, which should be proportional to the probability of host failure and inversely proportional to the amount of available resources. A dynamic replication scheme needs to be designed so that it can adjust to these factors. The case is more complicated when considering coordination among several clients: how to balance the amount of resources used and the turnaround of the tasks, so that all clients are satisfied.

Checkpointing is also a promising approach: the client's program states can be periodically stored for recovery and continued execution when there is failure. In clusters and institutional load sharing systems, checkpointing is done to a centralized and stable server. In a peer-based desktop grid system, the challenges include which

and how many peers should serve as checkpointing storage nodes, how to balance the checkpointing overhead and recovery speed with fast turnaround scheduling, and frequency of checkpointing. These questions become even more challenging considering the heterogeneity of system: the hosts in the peer-based desktop grid system have a variety of network connection speeds, clock speeds, and storage sizes.

A peer-to-peer checkpointing scheme can use the underlying DHT to periodically store the program states. DHTs have been used to build resilient distributed file systems, reputation systems, and for peer-based accounting. Development of a peer-to-peer checkpointing system using the power and elegance of DHT design and technology is an exciting new research direction.

### 7.2.3 Fairness in Peer-based Desktop Grid Systems

Peer-to-peer cycle sharing systems need to provide fairness to motivate nodes to contribute, and to provide good turnaround time for all clients. However, fairness is much harder to realize in an open dynamic network formed by anonymous nodes. While traditional systems use a central server to perform strict account management (users may need to acquire the account from some authority and then the activities of the users are monitored by that authority), the open peer-to-peer systems relax account management and then suffer from free riders, peers who take advantage of the system without donating anything to the system. How to prevent a resource hog from consuming a disproportionate share of the cycles?

Credit-based systems can be used as a solution, but there are many challenges in building such a system. First, how to evaluate the services of the hosts? Is one hour of contributed cycles on a fast host equal to one hour on a slow host? Second, how can credits can be used in scheduling? Strict credit-based scheduling is not enough in this case, as hosts may want to give higher priority to those who have directly served them even if other clients have more credits. Finally, how to secure this system against all sorts of attacks? For example, how to provide proof of service for the host's con-

tributed work and prevent the host from faking service? A possible solution to the former problem can be a signed acknowledgment issued by the client in exchange for keys to the encrypted results. The solution to the latter relies on result verification schemes. Our overall goal is to combine all sorts of services in deciding the fairness of the system. We believe that a comprehensive peer-based desktop grid system will soon be built to provide services not just limited to CPU cycles but also extended to storage and network bandwidth. Nodes should be rewarded for storing content, as well as contributing bandwidth and cycles. The design of a fairness scheme that takes all these different factors into consideration is an interesting and challenging problem.

Peer-based desktop grid systems provide compute cycles to a much broader range of users than could have been imagined just a few years ago. While scientists and mathematicians have utilized manual techniques and institutionally-based load sharing software like Condor for years, the average citizen is not yet aware that such an opportunity is possible. We expect many types of users to take advantage of the peer-based desktop grid systems. Independent artists can use these systems for scene rendering; chess hobbyists can test their latest programs; high school students can use open desktop grids for science fair or art fair projects. Such a diverse user population will be motivated to use cycle sharing if the research community can find ways to make the system fast, reliable, and secure. This dissertation addresses fundamental scientific questions regarding fast-turnaround scheduling in open peer-based desktop grid systems. Our research is a significant step in the direction of providing the best service to users in a open cycle sharing infrastructure.

## BIBLIOGRAPHY

- [1] ABERER, K., DATTA, A., HAUSWIRTH, M., AND SCHMIDT, R. Indexing data-oriented overlay networks. In *31st International Conference on Very Large Databases VLDB'05* (2005).
- [2] ANDERSON, D., COBB, J., KORPELA, E., LEBOFISKY, M., AND WERTHIMER, D. SETI@home: An experiment in public-resource computing. *Communications of the ACM* 45 (2002).
- [3] ANDRADE, N., CIRNE, W., BRASILEIRO, F., AND ROISENBERG, P. Our-Grid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing* (June 2003).
- [4] ANDRZEJAK, A., AND XU, Z. Scalable, efficient queries or grid information services. In *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing* (2002).
- [5] ANNEXSTEIN, F., AND BERMAN, K. "modeling peer-to-peer network topologies through "small-world" models and power laws". In *Proc. IX TELECOM-MUNICATIONS FORUM TELFOR 2001* (November 2001).
- [6] BARATLOO, B., KARAU, M., KEDEM, Z., AND WYCKOFF, P. Charlotte: Metecomputing on the web. In *Proceedings of the 9th International Conference on Paralel and Distributed Computing Systems* (1996).
- [7] BASU, S., BANERJEE, S., SHARMA, P., AND LEE, S. Nodewiz: Peer-to-peer resource discovery for grids. In *the 5th International Workshop on Global and Peer-to-Peer Computing (in conjunction with ccGrid 2005)* (May 2005).
- [8] BERMAN, F., HEY, A., AND FOX, G., Eds. *Grid Computing: Making The Global Infrastructure a Reality*. wiley, 2003.
- [9] BERMAN, F. High-performance schedulers. In *The GRID Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Ed. Morgan Kaufmann, 1999.
- [10] BREVIK, J., AND NURMI, D. Quantifying machine availability in networked and desktop grid systems. Tech. Rep. CS2003-37, UCSB, 2003.



- [11] BUTT, A., FANG, X., HU, Y., AND MIDKIFF, S. Java, peer-to-peer, and accountability: building blocks for distributed cycle sharing. In *Proceedings of the 3rd USENIX Virtual Machines Research and Technology Symposium (VM '04)* (May 2004).
- [12] BUTT, A., ZHANG, R., AND HU, Y. A self-organizing flock of condors. In *Proceedings of SC2003* (November 2003).
- [13] BUTT, A., ZHANG, R., AND HU, Y. A self-organizing flock of condors. *Journal of Parallel and Distributed Computing (JPDC)* 66, 1 (January 2006).
- [14] CALDER, B., CHIEN, A., WANG, J., AND YANG, D. The Entropia virtual machine for desktop grids. In *Proceedings of the First ACM/USENIX Conference on Virtual Execution Environments (VEE'05)* (2005).
- [15] CAMIEL, N., LONDON, S., NISAN, N., AND REGEV, O. The PopCorn project: Distributed computation over the Internet in java. In *Proceedings of The 6th International World Wide Web Conference* (1997).
- [16] CAPPELLO, F., DJILALI, S., FEDAK, G., HERAULT, T., MAGNIETTE, F., AND LODYGENSKY, V. Computing on large scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. In *FGCS Future Generation Computer Science* (2004).
- [17] CAPPELLO, F., FEDAK, G., MAGNIETTE, F., LODYGENSKY, O., AND FEDAK, G. XtremWeb: a global computing experimental platform <http://www.lri.fr/fedak/xtremweb/introduction.php3>.
- [18] CASANOVA, H., HAYES, J., KONDO, D., YANG, Y., RAADT, K., HE, J., AND LEGRAND, A. The Grid Research and Innovation Laboratory (GRAIL) in UCSD <http://grail.sdsc.edu/>.
- [19] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., AND ROWSTRON, A. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)* 20, 8 (2002).
- [20] CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM'03* (August 2003).
- [21] CHEN, L., CANDAN, K., TATEMURA, J., AGRAWAL, D., AND CAVENDISH, D. On overlay schemes to support point-in-range queries for scalable grid resource discovery. In *The Fifth IEEE International Conference on Peer-to-peer Computing* (2005).

- [22] CLARKE, I., MILLER, S., HONG, T., SANDBERG, O., AND WILEY, B. Protecting free expression online with freenet. *IEEE Internet Computing* 6, 1 (2002).
- [23] Clip2 distributed search services, the gnutella protocol specification (version 0.4, revision 1.2).
- [24] COHEN, B. Incentives build robustness in bittorrent. In *Proceedings of the first workshop on economics of peer-to-peer systems* (2003).
- [25] Collabnet inc., project jxta. <http://www.jxta.org/>.
- [26] COSTA, L., FEITOSA, L., ARAJO, E., MENDES, G., COELHO, R., CIRNE, W., AND FIREMAN, D. MyGrid: A complete solution for running bag-of-tasks applications. In *Proceedings of the SBRC 2004 - Salo de Ferramentas (22nd Brazilian Symposium on Computer Networks - III Special Tools Session* (May 2004).
- [27] CZAJKOWSKI, K., FOSTER, I., KESSELMAN, C., SANDER, V., AND TUECKE, S. Snap: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Proc. JSSPP'02* (2002).
- [28] DINDA, P. The statistical properties of host load. *Scientific Programming* 7, 3-4 (1999).
- [29] distributed.net, general-purpose distributed computing project, <http://distributed.net>.
- [30] DJILALI, S. P2P-RPC: programming scientific applications on peer-to-peer systems with remote procedure call. In *Proc. GP2PC2003 (Global and Peer-to-Peer Computing on Large Scale Distributed Systems) colocated with IEEE/ACM CCGRID* (2003).
- [31] EAGER, D., LAZOWSKA, E., AND ZAHORJAN, J. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. on Software Engineering* 12(5) (May 1986).
- [32] Entropia, inc., Entropia PC grid computing, <http://www.entropia.com>.
- [33] EPEMA, D., LIVNY, M., DANTZIG, R., EVERS, X., AND PRUYNE, J. A worldwide flock of condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems* 12 (1996).
- [34] ESTRIN, D., GOVINDAN, R., HEIDEMANN, J., AND KUMAR, S. Next century challenges: scalable coordination in sensor networks. In *MobiCom99* (1999).

- [35] FERREIRA, R., GRAMA, A., AND JAGANNATHAN, S. Enhancing locality in peer-to-peer networks. In *Proceedings of Tenth IEEE International Conference on Parallel and Distributed Systems* (July 2004).
- [36] FOR, G. AND PALLICKARA, S. NaradaBrokering: an event-based infrastructure for building scalable durable peer-to-peer Grids. In *Grid Computing Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2002.
- [37] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing* 11 (1997).
- [38] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.
- [39] FOSTER, I. AND IAMNITCHI, A. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)* (Feb 2003).
- [40] FOX, G., GANNON, D., KO, S., LEE, S., PALLICKARA, S., PIERCE, M., QIU, X., RAO, X., UYAR, A., WANG, M., AND WU, W. Peer-to-peer grids. In *Grid Computing Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2002.
- [41] FREY, J., TANNENBAUM, T., FOSTER, I., LIVNY, M., AND TUECKE, S. Condor-G: a computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)* (August 2001).
- [42] FU, Y., CHASE, J., CHUN, B., SCHWAB, S., AND VAHDAT, A. Sharp: an architecture for secure resource peering. In *SOSP 2003* (2003).
- [43] GANESAN, P., SUN, Q., AND GARCIA-MOLINA, H. Yappers: a peer-to-peer lookup service over arbitrary topology. In *IEEE Infocom'03* (2003).
- [44] GKANTSIDIS, C., MIHAIL, M., AND SABERI, A. Hybrid search schemes for unstructured peer-to-peer networks. In *Infocom'05* (2005).
- [45] GUPTA, R., AND SOMANI, A. CompuP2P: An architecture for sharing of computing resources in peer-to-peer networks with selfish nodes. In *Proceedings of second Workshop on the Economics of peer-to-peer systems* (2004).
- [46] HAN, J., AND PARK, D. A lightweight personal grid using a supernode network. In *Third International Conference on Peer-to-Peer Computing (P2P'03)* (2003).

- [47] HANDLER, G., AND MIRCHANDANI, P. *Location on Networks: Theory and Algorithms*. The MIT Press, 1979.
- [48] HAUSWIRTH, M., AND SCHMIDT, R. An overlay network for resource discovery in grids. In *Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'2005)* (2005).
- [49] HAYES, B. Computing science - collective wisdom. In *American Scientist* (March-April 1998).
- [50] HAYNES, T., HEDETNIEMI, S., AND SLATER, P. *Fundamentals of Domination in Graphs*. CRC Press, 1998.
- [51] HEINE, F., HOVESTADT, M., AND KAO, O. Towards ontology-driven p2p grid resource discovery. In *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)* (2004).
- [52] HUANG, Y., AND BHATTI, S. Decentralized resilient grid resource management overlay networks. In *Services Computing, 2004 IEEE International Conference on (SCC'04)* (2004).
- [53] IAMNITCHI, A., AND FOSTER, I. A peer-to-peer approach to resource location in grid environments. In *Grid Resource Management*, J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, Eds. Kluwer, 2003.
- [54] IYER, S., ROWSTRON, A., AND DRUSCHEL, P. Squirrel: A decentralized, peer-to-peer web cache. In *Proceedings of Principles of Distributed Computing (PODC 2002)* (2002).
- [55] KANG, X., ZHOU, D., RAO, D., LI, J., AND LO, V. Sequoia: A robust communication architecture for collaborative security monitoring systems. In *Poster session, SIGCOMM'04* (2004).
- [56] KLINGBERG, T., AND MANFREDI, R. The gnutella protocol specification (version 0.6), 2002.
- [57] KONDO, D., CASANOVA, H., WING, E., AND BERMAN, F. Models and scheduling mechanisms for global computing applications. In *International Parallel and Distributed Processing Symposium (IPDPS)* (2002).
- [58] KONDO, D., CHIEN, A., AND CASANOVA, H. Resource management for rapid application turnaround on enterprise desktop grids. In *ACM Conference on High Performance Computing and Networking, SC2004* (November 2004).

- [59] KONDO, D., TAUFER, M., BROOKSIII, C., CASANOVA, H., AND CHIEN, . A. Characterizing and evaluating desktop grids: An empirical study. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)* (2004).
- [60] KUMAR, A., XU, J., AND ZEGURA, E. Efficient and scalable query routing for unstructured peer-to-peer networks. In *Infocom'05* (2005).
- [61] LEDLIE, J., SHNEIDMAN, J., SELTZER, M., AND HUTH, J. Scooped, again. In *Proc. 2nd International Workshop on Peer-to-Peer systems (IPTPS'03)* (Berkeley,CA, Feb 2003).
- [62] LI, Y., AND MASCAGNI, M. Improving performance via computational replication on a large-scale computational grid. In *Third International Workshop on global and peer-to-peer computing* (2003).
- [63] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor -a hunter of idle workstations. In *the 8th International Conference on Distributed Computing Systems* (1988).
- [64] LIU, Y., LIU, X., XIAO, L., NI, L., AND ZHANG, X. Location-aware topology matching in p2p systems. In *IEEE Infocom'04* (2004).
- [65] LIVNY, M., AND RAMAN, R. High-throughput resource management. In *The GRID Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [66] LO, V., AND MACHE, J. Job scheduling for prime time vs. non-prime time. In *Proc 4th IEEE International Conference on Cluster Computing (CLUSTER 2002)* (Sep 2002).
- [67] LO, V., ZAPPALA, D., ZHOU, D., LIU, Y., AND ZHAO, S. Cluster Computing on the Fly: P2P scheduling of idle cycles in the Internet. In *IPTPS* (2004).
- [68] LO, V., ZHOU, D., LIU, Y., GAUTHIERDICKY, C., AND LI, J. Scalable supernode selection in peer-to-peer overlay networks. In *HotP2P'05* (July 2005).
- [69] LSF load sharing facility, <http://accl.grc.nasa.gov/lsf/aboutlsf.html>.
- [70] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. In *ICS'02* (2002).
- [71] LYNCH, A. *Distributed Algorithms*. Morgan Kaufmann, 1997.
- [72] MARZOLLA, M., MORDACCHINI, M., AND ORLANDO, S. Resource discovery in a dynamic grid environment. In *16th International Workshop on Database and Expert Systems Applications (DEXA'05)* (2005).

- [73] Miniwatts marketing group, internet world stats (usage and population statistic) <http://www.internetworldstats.com/stats.htm>.
- [74] NATRAJAN, A., NGUYEN-TUONG, A., HUMPHREY, M., HERRICK, M., CLARKE, B., AND GRIMSHAW, A. The legion grid portal. *Concurrency and Computation: Practice and Experience* 14, 13-15 (Grid Computing environments Special Issue) (2002).
- [75] Pande, V., and Stanford University, Folding@Home <http://folding.stanford.edu/>.
- [76] PlanetLab consortium, PlanetLab an open platform for developing, deploying and accessing planetary-scale services, <http://www.planet-lab.org/>.
- [77] RAMAN, R., LIVNY, M., AND SOLOMON, M. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)* (July 1998).
- [78] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content addressable network. In *Proc. ACM SIGCOMM* (Aug. 2001).
- [79] RATNASAMY, S., HANDLEY, M., KARP, R., AND SHENKER, S. Topologically-aware overlay construction and server selection. In *proceedings of IEEE INFOCOM'02* (2002).
- [80] RHEA, S., EATON, P., GEELS, D., WEATHERSPOON, H., ZHAO, B., AND KUBIATOWICZ, J. Pond: the oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)* (March 2003).
- [81] RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference'04* (2004).
- [82] RIPEANU, M., FOSTER, I., AND IAMNITCHI, A. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal special issue on peer-to-peer networking* 6, 1 (2002).
- [83] ROWSTRON, A., AND DRUSCHEL, P. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms* (Nov. 2001).
- [84] SETI@home: the search for extraterrestrial intelligence. <http://setiathome.ssl.berkeley.edu/>.

- [85] Sharman networks ltd, KaZaA <http://www.kazaa.com>.
- [86] SHEN, K., AND SUN, Y. Distributed hashtable on pre-structured overlay networks. In *Web Content Caching and Distribution WCW 2004* (2004).
- [87] SHIRAZI, B., HURSON, A., AND KAVI, K., Eds. *Scheduling and Load Balancing in Parallel and Distributed Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1995.
- [88] SHIVARATRI, N., KRUEGER, P., AND SINGHAL, M. Load distributing for locally distributed systems. *Computer 25(12)* (December 1992).
- [89] SINGLA, A., AND ROHRS, C. Ultrapeers: Another step towards gnutella scalability, 2002.
- [90] Space sciences laboratory, BOINC: Berkeley open infrastructure for network computing, <http://boinc.berkeley.edu/>.
- [91] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. Efficient content location using interest-based locality in peer-to-peer systems. In *IEEE Infocom'03* (2003).
- [92] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. Efficient content location using interest-based locality in peer-to-peer systems. In *IEEE Infocom* (2005).
- [93] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM* (Aug. 2001).
- [94] THAIN, D., TANNENBAUM, T., AND LIVNY, M. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. John Wiley & Sons Inc., 2002.
- [95] THAIN, D., TANNENBAUM, T., AND LIVNY, M. Condor and the grid. In *Grid Computing: Making The Global Infrastructure a Reality*, F. Berman, A. Hey, and G. Fox, Eds. wiley, 2003.
- [96] VAZIRANI, V. *Approximation Methods*. Springer-Verlag, 1999.
- [97] XU, Z., MAHALINGAM, M., AND KARLSSON, M. Turning heterogeneity into an advantage in overlay routing. In *IEEE Infocom'03* (2003).
- [98] XU, Z., TANG, C., AND ZHANG, Z. Building low-maintenance expressways for peer-to-peer systems. Tech. Rep. HPL-2002-41, Hewlett-Packard Labs, Palo Alto, CA, 2002.

- [99] YANG, B., VINOGRAD, P., AND GARCIA-MOLINA, H. Evaluating guess and non-forwarding peer-to-peer search. In *THE 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)* (2004).
- [100] YANG, L., FOSTER, I., AND SHOPF, J. Homeostatic and tendency-based CPU load predictions. In *Proceedings of IPDPS'03* (2003).
- [101] ZHANG, R., AND HU, Y. Assisted peer-to-peer search with partial indexing. In *Infocom'05* (2005).
- [102] ZHAO, B., HUANG, L., STRIBLING, J., RHEA, S., JOSEPH, A., AND KUBIATOWICZ, J. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications* 22, 1 (Jan. 2004).
- [103] ZHOU, D., AND LO, V. Cluster Computing on the Fly: resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of the 4th International Workshop on Global and P2P Computing (GP2PC'04)* (2004).
- [104] ZHOU, D., AND LO, V. Wave scheduler: Scheduling for faster turnaround time in peer-based desktop grid systems. In *Proc JSSPP'05* (2005).
- [105] ZHOU, D., AND LO, V. Wavegrid: a scalable fast-turnaround heterogeneous peer-based desktop grid system. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS'06)* (2006).
- [106] ZHOU, S., ZHENG, X., WANG, J., AND DELISLE, P. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience* 23, 12 (1993).



