Studies on Resource Management and Scheduling Schemes for Next Generation Distributed Applications

Hiroshi YAMAMOTO

Preface

The Internet is one of the largest communication networks consisting of a large number of computers. In the Internet, users can satisfy their objectives through many kinds of applications (e.g., electric mail, file transfer, etc.), and almost applications are working in a Client/Server-model where only specific high-performance computers, called *server*, handle all resource requests from other computers, called *client*. With the recent improvement in the performance of end-computers and networks, it is now feasible to construct a distributed environment where all end-computers in the Internet can serve as resource providers, and the users gather the resources needed for their objectives from the environment. Typical examples of applications based on it are Grid computing and P2P application. The distributed application can provide non-trivial services by gathering resources from the distributed environment, and balance the load between end-computers without concentrating load on specific computers.

In order to achieve high performance of distributed applications, a resource management architecture which helps users to effectively discover the resources and a scheduling algorithm which can efficiently utilize the resources are needed. The Grid computing and P2P application construct the resource management architecture in a different way, and provide the uniform interface for accessing resources to users. In addition, a resource scheduling algorithm, especially for the Grid computing, can efficiently utilize the resources and achieve a high-performance application processing. However, the existing resource management architecture and scheduling algorithm have following problems. In the distributed environ-

ment, the resources are managed in a distributed manner, so that the resource information obtained by the users has some uncertainty. In addition, since the users selfishly select resources for their objectives, the resources are unfairly utilized. These factors may concentrate the load on specific resources. Furthermore, the existing scheduling algorithms do not focus on a realistic heterogeneous environment in terms of the resource capacities of computers. Therefore, they cannot derive the optimal parameters for the application processing in the heterogeneous environment and cannot fully utilize the resource.

The objective of this dissertation is to modify/extend the existing resource management architecture and scheduling algorithm so as to solve the problems mentioned above, and achieve the high-performance processing of the distributed application.

First, in Chapter 2, I present the existing resource management for the P2P application and the Grid computing. Furthermore, the existing resource scheduling algorithms which achieve the efficient application processing are shown.

Second, in Chapter 3, I focus on the resource management architecture for the P2P application, and attempt to balance the load on storage between computers (or peers) dispersed over the P2P network. In the P2P network, the number of adjacent peers (degree) of each peer follows the power-law, and a resource request query traverses high-degree peers with high probability. Therefore, by allocating replicas of the original data on the high-degree peers, a replication method improves the search performance. However, the existing replication method concentrates the resource requests on a small number of high-degree peers storing a large number of replicas, which biases the storage load between peers. Therefore, I propose new replication methods for balancing the storage load between peers while limiting the degradation of the search performance within an acceptable level, and show their effectiveness in balancing the load through computer simulations.

Third, in Chapter 4, I focus on the impact of the characteristics of the resource management architecture for the Grid computing on a resource selection for application tasks. Since the resource management architecture for the Grid environment manages the resources in

the Grid environment in a distributed manner, it is essential that the resources are unevenly utilized and the utilization information of resources have some uncertainty. Therefore, when the scheduler selects resources for application processing, the characteristics of the resource management architecture should be considered. In this study, I propose some task allocation schemes (how to select resources for application tasks) with the consideration of the CPU utilization, and analytically evaluate the impact of the characteristics of the resource management architecture on the resource selection by modeling computers as an M/G/1-PS (Processor Sharing) queue. In addition, I investigate the performance of the task allocation schemes in a realistic Grid environment where the all computers composing the environment act as "users" through the computer simulation. As a result, I clarify the characteristics needed for the task allocation scheme adaptive to the Grid computing.

Fourth, in Chapter 5, I focus on the scheduling algorithm for the Grid computing, and aim at high-performance application processing by efficiently utilizing the resources gathered for the execution of applications. An existing scheduling algorithm, called Uniform Multi-Round (UMR) algorithm, has achieved good application turnaround time by processing a large amount of data in multiple rounds and overlapping the communication time and the computation time effectively. However, it does not focus on realistic heterogeneous environments where the network resources of all computers are different each other. Therefore, I propose a new scheduling algorithm, Parallel Transferable Uniform Multi-Round (PTUMR) algorithm, so as to minimize the application turnaround time, and show its performance through computer simulations. As a result, the proposed algorithm can dramatically mitigate the adverse effects of data transmissions of a large amount of application data, achieving turnaround time close to the theoretical lower limits.

Finally, I hope that this dissertation will be helpful for further study in this field.

Mar. 2006

Hiroshi YAMAMOTO

Acknowledgements

First of all, I wish to express my sincere appreciation to Professor Yuji Oie at Kyushu Institute of Technology. His constant encouragement, guidance through this research, invaluable discussions and advices have greatly helped in accomplishing the research. I also thank him for his careful reading of all papers on the research.

I wish to thank Professor Takaichi Yoshida at Kyushu Institute of Technology for his invaluable discussion and comments.

I also would like to express my gratitude to Associate Professor Kenji Kawahara, Associate Professor Masato Tsuru, Associate Professor Takeshi Ikenaga and Associate Professor Hiroshi Koide at Kyushu Institute of Technology and Tetsuya Takine at Osaka University for their valuable comments, time, and help in completing the research.

I wish to thank Mr. Daisuke Maruta now in NRI, Mr. Shigenori Nishi now in NTT DATA and Mr. Toshiya Okamura for their work, help and comments.

I also wish to thank Dr. Hiroyuki Koga at National Institute of Information and Communications Technology, Assistant Professor Yutaka Fukuda, Dr. Hitomi Tamura and Dr. Kei Ohnishi at Kyushu Institute of Technology for their advices and encouragement.

I extend thank Mr. Kazuya Tsukamoto, Ms. Manami Takemura, Ms. Chie Yoshiki and all other members of the Information Network Research Group at Kyushu Institute of Technology for their kind support and valuable discussions.

Finally, my greatest appreciation goes to my parents, Kozo and Kimiko Yamamoto for their understandings and support for me. And, I extend thank to Mr. Takeshi Ueda, Mr.

Kunitoshi Okamoto and all other friends for their warm encouragement and supports.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is one of the largest networks consisting of millions of computers. In the Internet, users can achieve their objectives through many kinds of applications which utilize resources dispersed over the Internet. Many applications running in the Internet (Web surfing, E-mail, etc.) are modeled in a Client/Server model where a small number of high-performance computers, called *server*, serve as a resource provider for others, called *client*. However, this model has a following problem. All resource requests are handled by only a small number of servers, so that, if they fail, the entire service of the application is stopped.

As the Internet widely spreads over a large number of end-computers, new type of applications such as Peer-to-peer (P2P) applications and Grid computing has emerged. These applications target a distributed environment where all computers can serve as the resource provider, and the users attempt to collect the resources required for their objectives from the distributed environment, while the traditional application adopts the centralized architecture (i.e., Client/Server model). In the distributed environment, the distribute application can continue to provide the services even if some computers fails. Therefore, I can say that the distributed application can solve problems existing in the Client/Server model.

In order to achieve the high-performance application processing, both the resource management architecture which helps the users to efficiently discover the resources and the scheduling algorithm which leads to an efficient utilization of the discovered resources are needed.

This chapter is organized as follows. In Section 1.1, I introduce the emergence of the next generation distributed applications (P2P application and Grid computing) due to the improvement in the performance of end-computers and networks. Next, in Section 1.2, I present the resource management architecture and the scheduling algorithm for the distributed application, and the problem of them is shown in Section 1.3. Finally, I show the outline of this dissertation in Section 1.4.

1.1 Emergence of next generation distributed applications

In the traditional application such as Web surfing and E-mail, the users attempt to access only a small number of high-performance computers (i.e., *servers*) to achieve their objectives, and almost all computers connecting to the Internet (i.e., *clients*) have had much lower capacities than the servers. With the recent improvement in the performance of end-computers, the clients becomes to have enough capacity to serve as resource providers, which make it possible to provide distributed applications such as Grid computing and Peer-to-peer (P2P) application in the Internet. The distributed application targets a distributed environment where all end-computers provide their resources for the application processing and users attempt to collect the necessary resource from them.

In this context, many technologies for the Grid computing and the P2P application have been developed respectively, and the resources for them are managed in a different manner. The technology for the Grid computing first developed as a middleware which constructs a hardware and software infrastructure by connecting computation resources through the Internet. The infrastructure provides dependable, consistent, pervasive, and inexpensive access to high-performance computation capacities [FK98a]. Recently, Grid technology attempts to build a fundamental infrastructure where various entities (e.g., computational resources, storages, sensors, etc.) can seamlessly collaborate for providing non-trivial qualities of services to users [FKT01]. In order to promote the collaboration, many standards organizations including the Global Grid Forum (GGF) are making many documents of standards for the Grid technologies [GGF]. On the other hand, the technology for P2P application firstly aims at a construction of an environment where the users can directly exchange their data each other and the storage resources of end-computers can be shared by participants of the P2P application. This technology helps making many file-sharing applications such as Napster [Nap] and Gnutella [Gnu]. Recently, new P2P technology attempts to provide an environment where the users' requests can smoothly meet the resources and services required for the users' requirements [SAZ$^+$04, STI02].

As I mentioned above, the technologies for the Grid computing and the P2P applications are built on the same objective, namely the construction of the communication infrastructure which promotes the collaboration of many kinds of resources and services, and each technology attempts to complement the function by considering the other [Bha05].

1.2 Resource management architecture and scheduling algorithm for distributed applications

In the distributed applications, the resource management architecture which enables the users to efficiently discover the resources (computation resource, storage, sensor, etc) is needed. The traditional IP network adopts Domain Name System (DNS) to find the location information (i.e., IP address) of the computer corresponding to the user-friendly name, but does not support a lookup mechanism for each resource in computers. The resource management architectures for the Grid computing and the P2P application have been proposed so far.

In the Grid environment, Grid middleware equipped in computers works as resource management architecture. It firstly constructs a virtual organization where the users and end-computers sharing the same objective, and then the resource information is managed by computers belonging to the same organization. In a de facto standard Grid middleware, Globus Toolkit [Fos05], developed by the Globus Alliance [Glo], the information service called Monitoring and Discovery Service (MDS) is implemented [MDS], which achieves scalable, uniform and efficient access to distributed resources. Each computer running MDS mon-

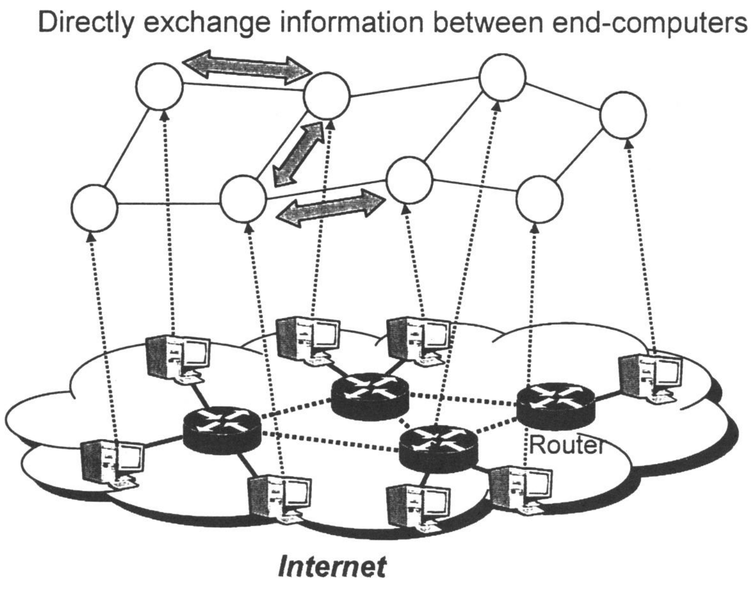

Figure 1.1: Resource Management Architecture for P2P application.

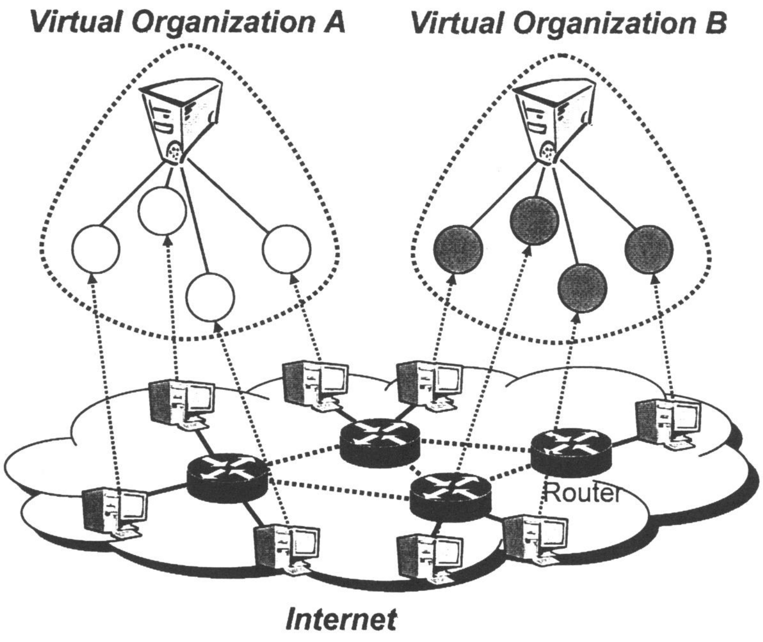

Figure 1.2: Resource management architecture for Grid computing.

itors the resources periodically and registers the resource information to its own database. In addition, an index server (which usually exists in each domain) manages a presence of computers running MDS, and when receiving a resource request, it asks the computers to send back their resource information. The resources are managed with respect to each virtual organization, and all index servers in the organization collaborate to provide a uniform interface to the users. By sending the resource request to any server, the users can obtain any resource information in the organization.

On the other hand, in the P2P technology, each end-computer (or *peer*) has both client and server function, and the resource information is managed in a completely distributed manner. Each peer connects to some other peers by establishing TCP connections. By propagating the request queries over the P2P network through the virtual links between peers, the users can discover the requested resources (the way to discover resources is categorized into several types as shown in Section 2.1.1). Furthermore, the P2P technology has a replication function which allocates replicas of the data owned by some peers to other sites. Due to the replication, the users can efficiently discover the requested resource by accessing the nearest peer storing the replica instead of the peer storing the original data, and the load on the storage resource storing the original data can be distributed over the entire P2P network.

Furthermore, in order to achieve the efficient application processing, we should optimize how to use the resources gathered for the users' objectives. In the Grid computing, resources are utilized by adopting a Master/Worker model where the computer (Master) having application tasks and data dispatches subtasks to other computers (Worker). A typical example of applications based on this model is Divisible Workload Application, where the application data can be divided into an arbitrary number of chunks with an arbitrary size, and the computation time and communication time of chunks are proportional to the size of them. By focusing on this algorithm, some scheduling algorithms which attempt to minimize the application turnaround time have been proposed so far [BGMR96, Rob03].

1.3 Problems in distributed applications

As I mentioned in the previous section, in order to achieve the efficient processing of the distributed applications, the resource management architecture and the scheduling algorithm have been proposed so far. However, existing proposals have some problems.

First, the existing architecture is insufficient to achieve a load-balancing between computers, and the load concentrates on a small number of computers. In the Grid environment, the resources are managed in a distributed manner with respect to the virtual organization, and the resource information is updated periodically. Here, the scheduler in each organization handles application tasks generated by the users belonging to the same organization. In a condition where resources can belong to multiple virtual organizations, each scheduler cannot obtain the latest resource information and cannot perceive task allocations of other schedulers. Therefore, the scheduler overestimates the resource states of the high-performance resources, which may bias the load between computers. On the other hand, although the replication for the P2P application helps balancing the load between peers by distributing the replicas of the data over the entire P2P network, the data replications and requests from users tend to concentrate on a small number of computers with high-degree. This is because the P2P network has a characteristic that the number of adjacent computers (degree) of each computer follows the power-law network, and the resource request queries go through a small number of high-degree peers with high probability. Therefore, high-degree peers play an important role in the P2P application, and the failure of them may markedly degrade the performance of applications.

On the other hand, the existing scheduling algorithm for the Divisible Workload application can effectively utilize the resources discovered for the user's objective, but focus on a sequential transmission model which allows the master to transmit application data to one worker at a time. It cannot consider a realistic Grid environment where there exist a difference between the data transmission speed of the master-side link and that of the worker-side link, while it can handle the heterogeneous environment in terms of the worker's computation/communication capacities. Therefore, when the master-side link capacity is larger than the worker-side link capacity, the existing algorithm cannot fully utilize the master-side link capacity, and it cannot minimize the application turnaround time. Therefore, in order to achieve high-performance application processing, I should consider completely heterogeneous environments where all computers including both the master and workers have different communication/computation capacities.

1.4 Outline of this dissertation

I have introduced the emergence of the next generation distributed applications, and the resource management architecture and scheduling algorithm needed for them has been shown. In the distributed applications such as the Grid computing and the P2P application, it is necessary for the users to discover resources needed for their objectives, so that the technology for the Grid computing adopts the distributed database, and that for the P2P application adopts the completely distributed resource search method as resource management architecture. In addition, in order to achieve the high-performance application processing, some scheduling algorithms which efficiently utilize the resources has been proposed so far. However, the existing resource management architecture and the existing scheduling algorithm have some problems, namely the concentration of the load on a few resources and the ineffective utilization of the resources.

In this dissertation, I first propose the way to distribute the load over the entire distributed environment. In addition, by focusing on the Grid computing, I propose new scheduling algorithm adaptive to completely heterogeneous environment in terms of the computers' capacities.

First, in Chapter 2, I present the resource management architecture which helps the users to efficiently discover the necessary resources. In addition, I show the scheduling algorithm for the discovered resources by focusing on the Grid computing.

Second, in Chapter 3, I focus on the resource management architecture for the P2P application, and attempt to balance the load on storage between computers (or peers) dispersed over the P2P network. In the P2P network, the number of adjacent peers (degree) of each peer follows the power-law, and a resource request query traverses high-degree peers with high probability. Therefore, by allocating replicas of the original data on the high-degree peers, a replication method improves the search performance. However, the existing replication method concentrates the resource requests on a small number of high-degree peers storing a large number of replicas, and biases the storage load between peers. Therefore, I propose the replication methods for balancing the storage load between peers while limiting the degradation of the search performance within an acceptable level, and show their effectiveness in balancing the load through computer simulations.

Third, in Chapter 4, I focus on the impact of the characteristics of the resource management architecture for the Grid computing on a resource selection for application tasks. Since the resource management architecture for the Grid environment manages the resources in the Grid environment in a distributed manner, it is essential that the resources are unevenly utilized and the utilization information of resources have some uncertainty. Therefore, when the scheduler selects resources for application processing, the characteristics of the resource management architecture should be considered. In this study, I propose some task allocation schemes (how to select resources for application tasks) with the consideration of the CPU utilization, and analytically evaluate the impact of the characteristics of the resource management architecture on the resource selection by modeling computers as an M/G/1-PS (Processor Sharing) queue. In addition, I investigate the performance of the task allocation schemes in a realistic Grid environment where the all computers composing the environment act as "users" through the computer simulation. As a result, I clarify the characteristics needed for the task allocation scheme adaptive to the Grid computing.

Fourth, in Chapter 5, I focus on the scheduling algorithm for the Grid computing, and aim at high-performance application processing by efficiently utilizing the resources gathered for the execution of applications. An existing scheduling algorithm, called Uniform Multi-

Round (UMR) algorithm, has achieved good application turnaround time by processing a large amount of data in multiple rounds and overlapping the communication time and the computation time effectively. However, it cannot fully utilize the communication resources in the realistic heterogeneous environment where the master-side link capacity is different from the worker-side link capacity. Therefore, I propose a new scheduling algorithm, Parallel Transferable Uniform Multi-Round (PTUMR) algorithm, so as to improve the application turnaround time, and show its performance through computer simulations. As a result, the proposed algorithm can dramatically mitigate the adverse effects of data transmission of a large amount of application data, achieving turnaround time close to the theoretical lower limits.

The results discussed in Chapter 3 is mainly taken from [YKTO03, YKTO06], Chapter 4 from [YMO05, YMO06], and Chapter 5 from [YTO05].

Chapter 2

Resource Management Architecture for P2P application and Grid computing

In this chapter, I introduce the resource management architecture for the P2P application and the Grid computing which helps the users to efficiently discover the necessary resources. In addition, the scheduling algorithms which can effectively utilize the resources to maximize the performance of applications are shown.

2.1 P2P applications

The Napster and Gnutella have pioneered the idea of a Peer-to-Peer (P2P) file-sharing application. These applications provide the scalable and robust environment where millions of users can directly exchange the data, and thus share the storage resources in their PCs without intermediating centralized servers. The resource management of the file-sharing application can extend to other kinds of applications. For example, by adopting the technology for the file-sharing applications to a management of a telephone directory, Skype can provide Voice over IP (VoIP) service hosting millions of users simultaneously. As a result, the resource management architecture will extend to locating not only resources but also services and users.

Therefore, I can say that, in the future, the resource management for P2P applications will evolve into the fundamental technology which helps the seamless collaboration of the users, services and resources.

In the resource management architecture for the P2P application, the user issues the query for lookup of resources, services and other users. In the following section, we show the query forwarding method which is key technology of the lookup service.

2.1.1 Query forwarding method

In the P2P network, each computer (or peer) connects to some other peers through the TCP connections. The connection between peers is referred to as a virtual link, and the users send queries through the virtual links to locate the necessary data. The way to discover the requested resource can be roughly categorized into two types, Hybrid P2P model and Pure P2P model.

Hybrid P2P model

Hybrid P2P is a network model where there is a centralized directory server managing the location of the data in the P2P network. In this model, the user issue queries to the directory server to discover the peer storing the requested data, while the data is directly exchanged between peers. A typical example of applications based on this model is Napster [Nap]. Fig. 2.1 shows an example where Peer B want to retrieve the data stored in Peer A, but does not know where the requested data is placed. Peer B first requests the directory server to send back the location of the requested data. After receiving the location, Peer B directly connects to Peer A to get the data. In this model, lookup requests concentrates on the directory server, so that, the failure of the server stops the entire service of the application. Therefore, it is not adaptive to the environment hosting a large number of users.

Pure P2P model

Pure P2P model have no centralized directory server and consists of only end-computers. In this model, the data request queries are exchanged between peers without accessing the cen-

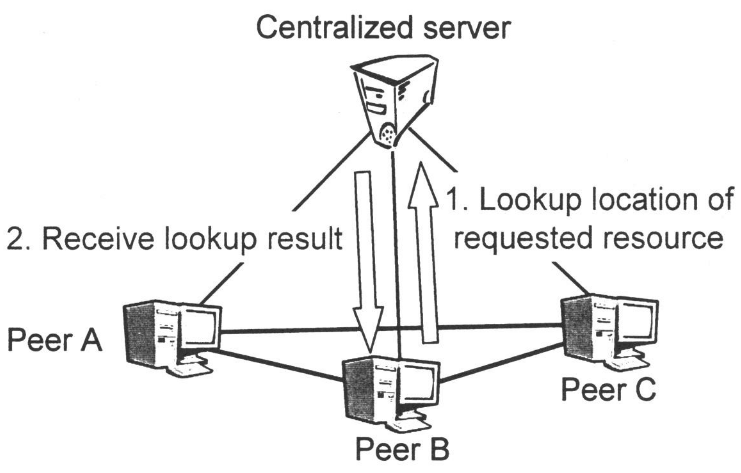

Figure 2.1: Lookup of requested data in Hybrid P2P.

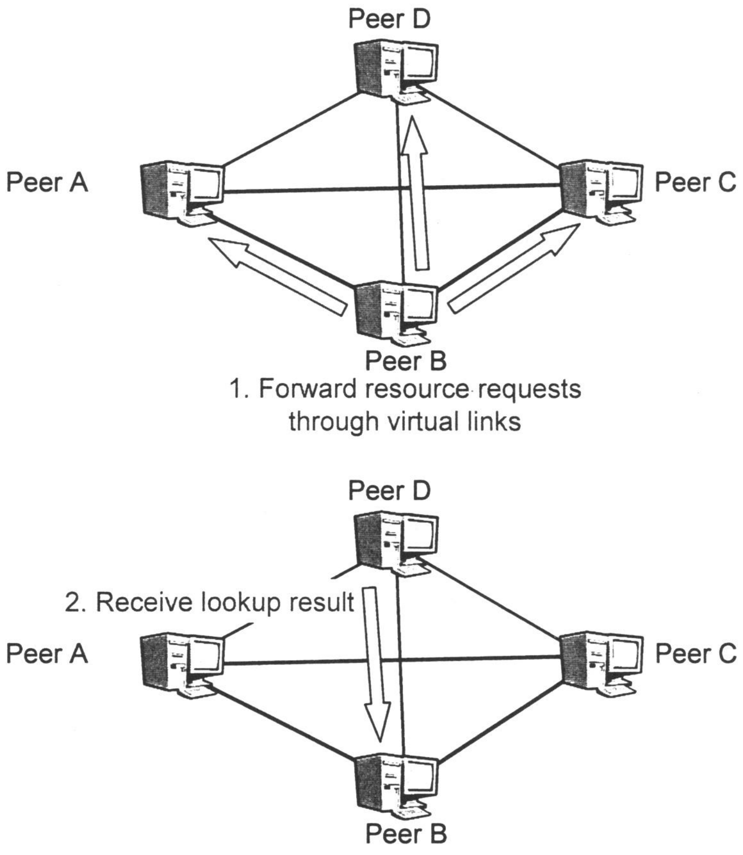

Figure 2.2: Lookup of requested data in Pure P2P.

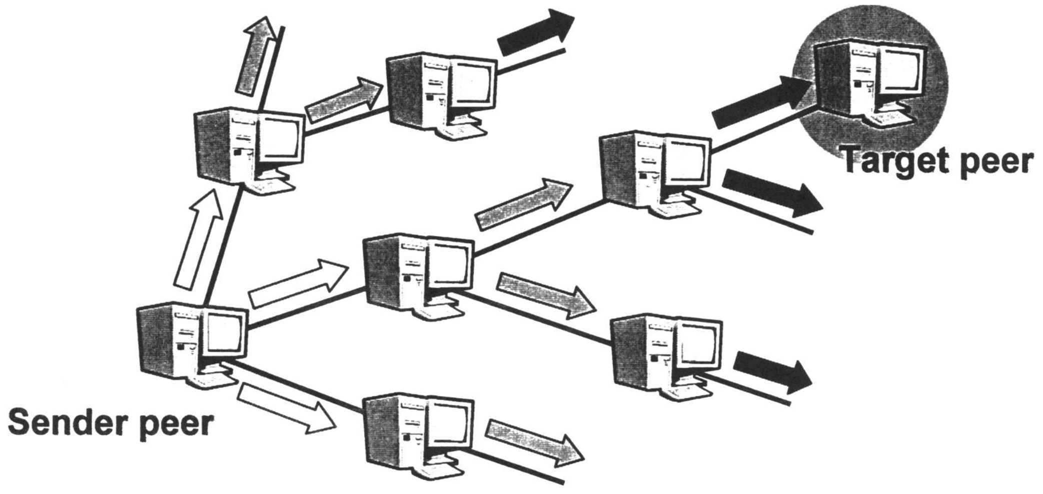

Figure 2.3: Query forwarding in Unstructured P2P network.

tralized server, that is, the users can locate the requested data in completely distributed manner. Typical examples of applications based on this model are Gnutella [Gnu] and Freenet [CSWH00]. Fig. 2.2 show the condition where Peer B requests the data stored in Peer D. First Peer B records its address and a name of the requested data in the query and propagates it through the virtual links between peers. When Peer D, which stores the requested data, receives the query, it sends back the location information of the data to the source address recorded in the query. After that, Peer B directly accesses Peer D and gets the requested data. This model scales a large number of users because it does not have one point of failure and can well balance the load of the data request and the data transmission over the P2P network.

The Pure P2P model is categorized into two types, namely Unstructured and Structure, and they construct different of networks.

Unstructured P2P network

Unstructured P2P network is formed by peers joining the P2P network following some loose rules. This network has no strict control to construct the network topology, and each peer maintains the virtual links connected to some other peers as in Fig. 2.3. As shown in this

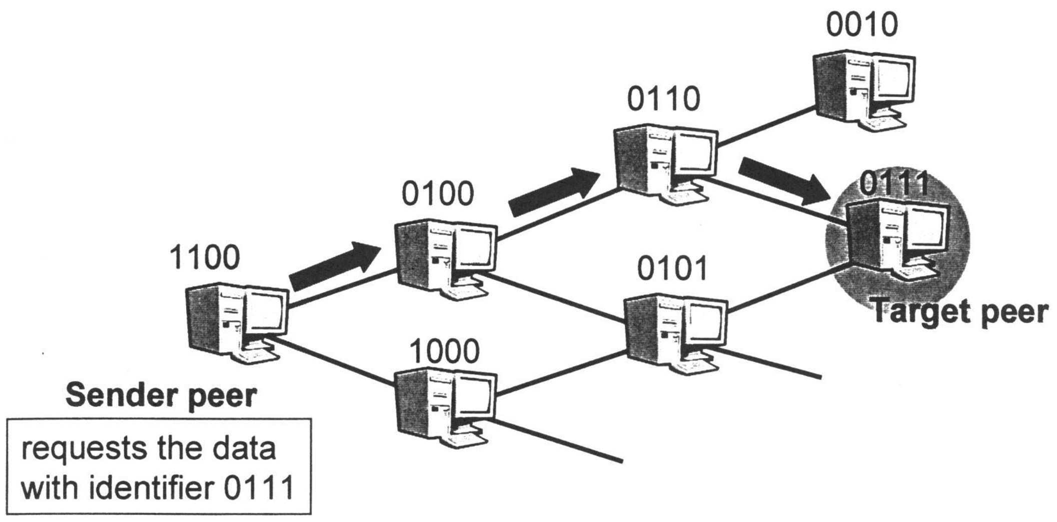

Figure 2.4: Query forwarding in Structured P2P network.

figure, when requesting the data, the peer generates the query, records the address and the name of the requested data into it, and sends it to some adjacent peers. Each peer receiving the query evaluates the query locally, and if it stores the requested data, it sends the data to the source address recorded in the query. Otherwise, the peer forwards the query to some adjacent peers recursively. The process is repeated until the requested data is discovered. Key issue of this model is the way to determine where each peer forwards the query in each step, and there are some proposals at this point (described in Section 2.3.1) [LCC+02, CMS04].

However, the lookup in the unstructured P2P network cannot reliably find the original data unless the flooded query reaches all peers. On the other hand, it is suitable to lookup of well replicated data [LRS02, CRB+03]. Therefore, the query forwarding method should work with appropriate replication methods (shown in Section 2.1.2).

Structured P2P network

Structured P2P network has strict rule to construct the network topology, and the location of the data is controlled [SMK+01, RFHK01, RD01a, ZHS+04]. In this network, an identifier is assigned to each data based on some information (e.g., file name), and each peer has also its own identifier. And then, a location information of the data is placed at the peer whose identifier is the nearest to the data's one. For example, Chord presented in [SMK$^+$01], the location information of the data is managed by the peer whose identifier is equal to or follows the identifier of the data. In addition, the way to forward queries is also well defined, and each peer manages the routing table and each entry of it includes a couple of peer's identifier and IP address. As shown in Fig. 2.4, when requesting the data, the peer issues the query to the peer whose identifier is nearest to the identifier of the requested data. This process is repeated until the query arrives at the peer managing the data's identifier.

The query forwarding method in the structure P2P network can reliably find the requested data in small number of search hops. However, this network has some problems. First, many message exchanges are necessary for each peer to keep entries of the routing table latest state. Especially when the frequent peer arrivals and departures occur, the routing table becomes unstable and cannot provide the efficient routing [RGRK04]. In addition, the implementation of functions in the structured P2P network such as a lookup of the requested data is much more difficult than that in the unstructured P2P network.

As I mentioned above, the query forwarding method in the unstructured P2P network overcomes that in the structured P2P network in terms of the search performance for the well replicated data, so that, in this dissertation, I focus on the unstructured P2P network, and discuss about the appropriate replication method to it.

2.1.2 Replication Method

Replication method determines where the replicas of the original data are placed. Due to the replication method, users can always get necessary data in the P2P network where the continuous peer arrival and departure occur. In addition, the replication method improves the search performance by allowing the users to access the nearest replica of the requested data.

In order to optimize the search performance, the replication method should satisfy a rule described in the following section.

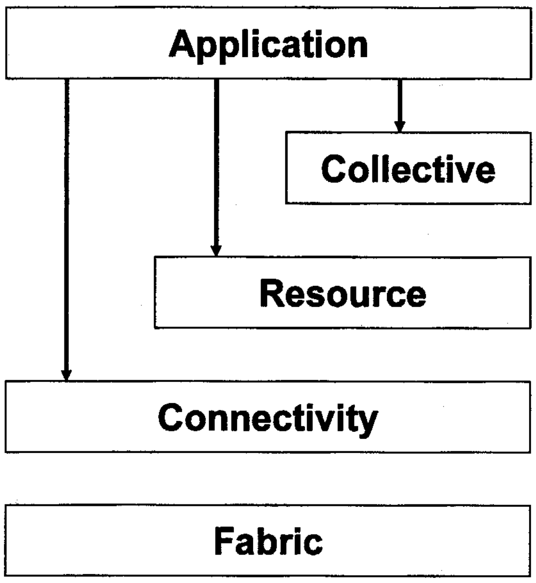

Figure 2.5: Grid Protocol Architecture.

Square-Root Replication

Existing research has clarified that the replicas of the original data should be allocated according to Square-Root manner to achieve the optimal search performance [LCC+02, CS02]. In the Square-Root replication, each data i is replicated at r_i random sites among all R sites. Here, r_i is derived by the following equation.

$$r_i = \frac{R \times \sqrt{q_i}}{\sum_i \sqrt{q_i}}. \tag{2.1}$$

where q_i indicates a relative rate at which the data i is requested among all data.

Surprisingly, the Square-Root replication can be achieved by simple distributed replication methods. I show an example of them, called Path Replication, in Section 2.3.1.

2.2 Grid computing

The objective of the Grid computing was to construct the high-performance computing environment by connecting the computers dispersed over the Internet and to provide an interface by which the users can uniformly and effectively access to the high-performance computation resources. Recently, it extends to providing an interface for many kinds of resources and equipments (e.g., computation/communication resources, sensors, etc.).

Figure 2.5 presents the grid protocol architecture which virtualizes resources dispersed over the Internet and provides the uniform interface to users [FKT01]. Like in IP protocol stack, functions in each layer are based on the characteristics and functions in the lower layer.

Fabric Layer

Fabric layer provides an interface for each local resource composing the grid environment (e.g., CPU, memory, storage, network) to the upper layer, and includes functions for effectively utilizing the local resource such as an advance reservation and scheduling.

Connectivity Layer

Connectivity layer includes core communication and authentication protocols. Communication protocols enable the exchange of data between resources through the interfaces defined in the Fabric Layer. Authentication protocols build secure communication to the resources by verifying the identity of users and resources.

Resource Layer

Resource layer builds on the communication and authentication protocols of the Connectivity layer and provides protocols for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Through the interface of the Resource layer, the upper layer functions or applications can handle each resource controlled by the Fabric layer.

The Resource layer and Connectivity layer achieve the virtualization of individual resource dispersed over the Internet, and allow the users to access all resources in the grid environment as if they use the local resources.

Collective Layer

While the Resource layer focuses on interactions with a single resource, Collective layer supports the protocols for interactions across collections of resources. Collective layer build on functions of the lower layers (Resource and Connectivity layers), and achieve a wide range of resource sharing without depending on other protocols.

An example of services based on the Collective layer is a directory service which allows the users to locate the resources satisfying their requirements.

Application Layer

Application layer comprises the user applications executing in the grid environment. The user application utilizes the necessary functions by accessing the interfaces provided in the lower layers for the objective.

In order to provide the Grid computing, the functions and interfaces in each layer should be developed. Globus toolkit is a de facto standard middleware constructed based on the layered architecture.

2.2.1 Globus Toolkit

The Globus toolkit produced by the Globus Alliance [Glo] is a software providing the indispensable functions for the Grid computing. One of main functions in the Globus toolkit is a resource management which helps the user application to discover the resources and allocates the grid jobs to the resources.

In this section, I present two services, Monitoring and Discovery Service (MDS) and Grid Resource Allocation Manager (GRAM), composing of the resource management architecture in the grid environment.

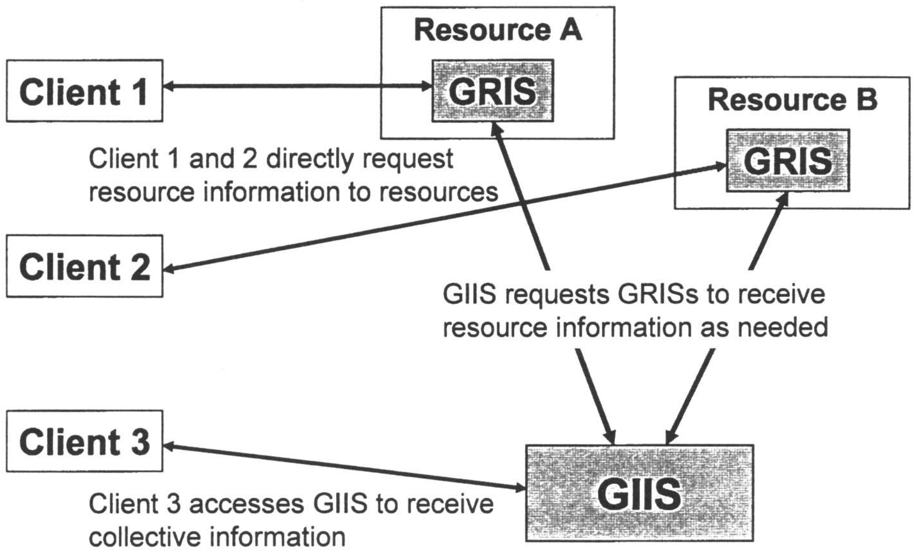
Resource A
GRIS
Client 1
Resource B
GRIS
Client 1 and 2 directly request resource information to resources
Client 2
GIIS requests GRISs to receive resource information as needed
Client 3
Client 3 accesses GIIS to receive collective information
GIIS

Figure 2.6: Architecture of MDS.

Monitoring and Discovery Service (MDS)

MDS allows the users to discover the resource information in the Grid environment [CFFK01, MDS]. It consists of two sub-services, Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS) as shown in Fig. 2.6. GRIS is a common information service running in each resource, and GIIS discovers the information required by the users.

Grid Resource Information Service (GRIS)

GRIS monitors the resource information of each computer including static information (e.g., operating system version, CPU type, etc) and dynamic information (e.g., CPU utilization, storage utilization, etc), and notifies the information to the index server (GIIS).

GRIS authenticates and parses each incoming request and then dispatches them to local information providers. One information provider exists in each resource, and is responsible for monitoring the resource information. After receiving the requested information from the information providers, GRIS aggregates them to the requested form and sends it to the index server.

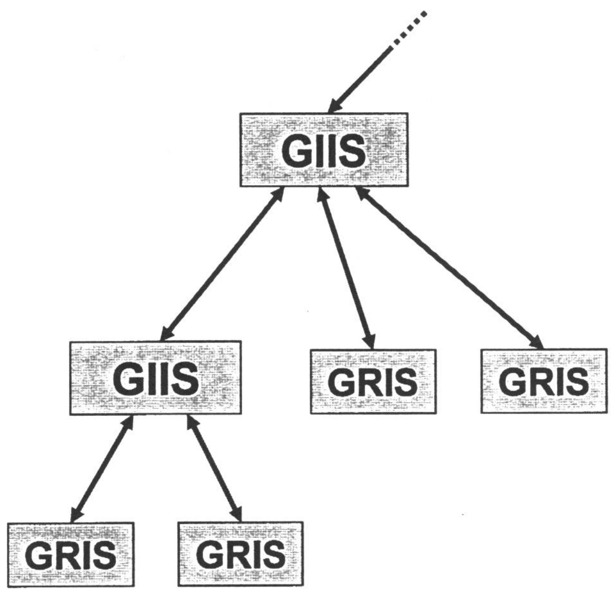

Figure 2.7: Hierarchical Architecture of MDS.

Grid Index Information Service (GIIS)

GIIS provides a framework which allows the index servers to construct a hierarchical structure. One or more GRISs connect to one GIIS server, and GIIS server can also connect to other GIIS server. The GIIS server periodically receives notification messages from GRISs and GIIS server connecting to it, and manages the indices of them. One hierarchical architecture shown in Fig. 2.7 corresponds to one virtual organization, and the user issues the requests to the GIIS server to get the resource information in the organization. After receiving the request, the GIIS server requests GRISs to send back the information, aggregates the information to the requested form and send it to the user.

Grid Resource Allocation Manager (GRAM)

GRAM defines the interface which enables the users to execute their jobs on the local and/or remote resources. This interface also includes the functions for monitoring and terminating the job.

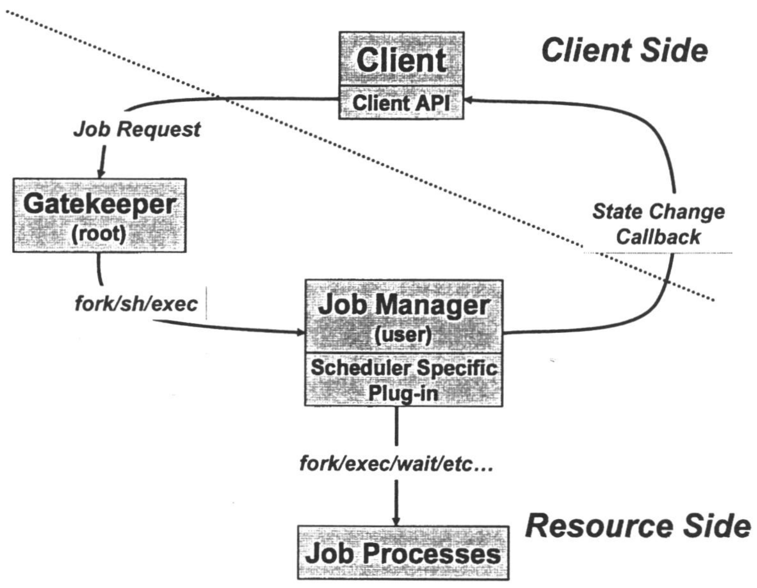

Figure 2.8: Components in GRAM.

GRAM consists of several components described in Fig. 2.8. In order to submit jobs, the user first issues the request to the *Gatekeeper* on the local and/or remote computer. The Gatekeeper parses the request and starts the *Job manager* for each job. The Job manager is responsible for executing and monitoring the job, and notifying the processing state of the job to the user. When the job is completed, the Job manager expires.

2.2.2 Scheduling method

Scheduling is defined as the process of mapping the user jobs to resources in a way that the user and application requirements (e.g., application turnaround time) is satisfied, and there exist several scheduling strategies. In this section, I presents two strategies, centralized scheduling and distributed scheduling [HSSY00].

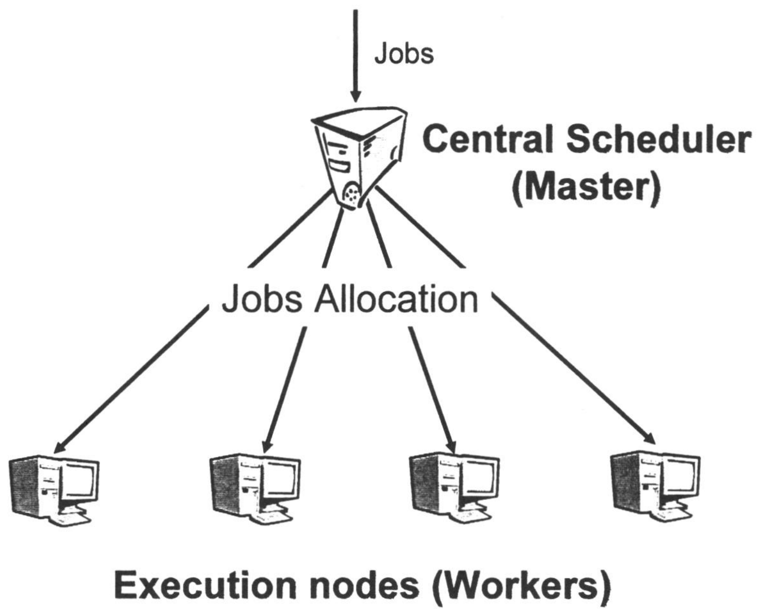

Figure 2.9: Centalized Scheduling.

Centralized Scheduling

In a centralized scheduling environment, a central scheduler serves as a resource manager to schedule jobs to other execution nodes as shown in Fig. 2.9. In this strategy, jobs are first submitted to the central scheduler, and it then allocates jobs to appropriate execution nodes by considering the requirements of the users and/or applications. This model is also referred to as a master/worker model where one master (central scheduler) manages its workers (execution nodes).

The centralized scheduling strategy is suitable to the environment where the central scheduler can get the up-to-date resource information of all execution nodes, because it can make the best decision by considering the resource capacity and availability. However, since the central scheduler may become a bottleneck, the centralized scheduling strategy does not scale well with the increasing the size of environment.

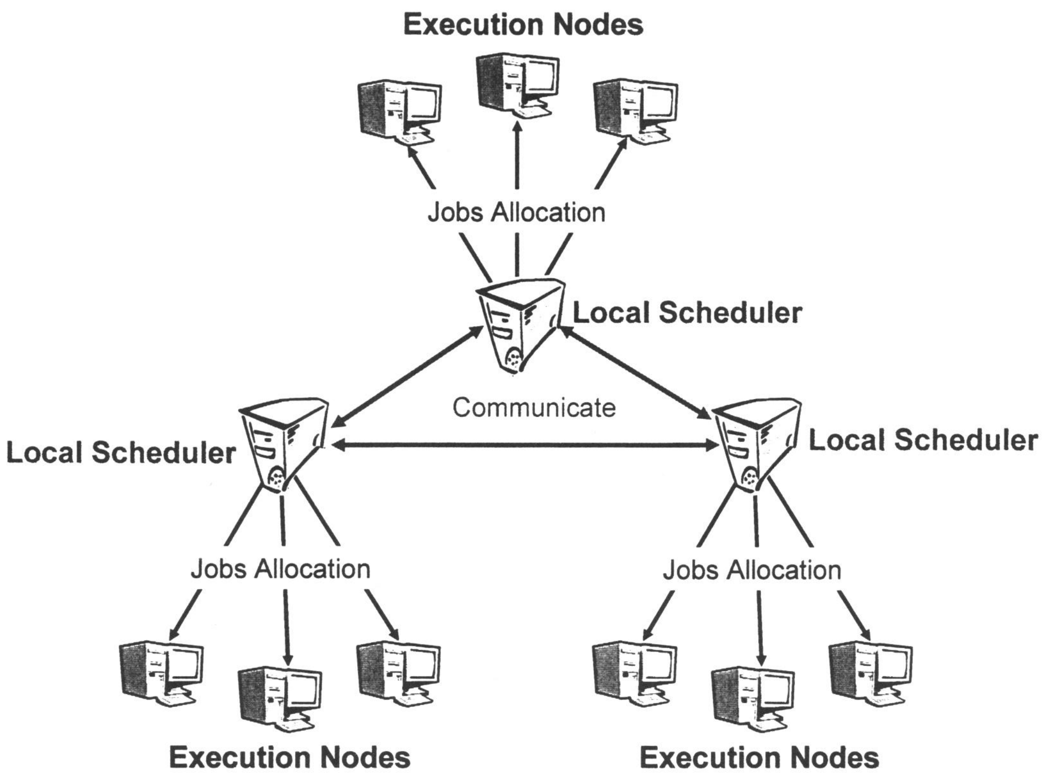

Figure 2.10: Distributed Scheduling.

Distributed Scheduling

In Distributed Scheduling strategy, there is no centralized scheduler managing all nodes. Instead, the environment includes multiple local schedulers, and each local scheduler is responsible for one group of execution nodes (e.g., virtual organization). The local scheduler has a list of remote schedulers, and when dispatching jobs to remote groups, it directly communicates to the remote scheduler, and issues the request of the job allocation.

The Distributed scheduling overcomes the centralized scheduling in terms of scalability because there is no bottleneck in the environment. However, each local scheduler cannot manage the up-to-date utilization information of all execution nodes, so that, the distributed scheduling model usually leads to sub-optimal decision.

2.3 Related Work

In this section, I describe related works. First, I focus on the improvement of two performance metrics in the P2P application, and present the existing proposals. And then, the existing scheduling algorithms which achieve the high performance of the Grid computing are shown.

2.3.1 Search performance and Load balancing in P2P application

Many studies on the P2P network consider not only the search performance but also the load balancing between peers at the viewpoint where the peers with the same functions should be equal in load. The structured P2P network and unstructured P2P network attack these problems in the different way.

Structured P2P network

In order to improve the search performance for the requested data and the performance in load balancing, most researches have focused on the structured P2P network using Distributed Hash Table (DHT) [SMK+01, RFHK01, RD01a, ZHS+04]. Identifiers of each data and peer described in Section 2.1.1 are derived by inputting some information (e.g., file name of data, IP address of peer, etc.) into a hash function, and the location information of the data is place at the peer whose identifier is the closest to the data's identifier. Each peer manages the routing table, and each entry in it contains a pair of identifier of data and an IP address of peer. By referring the routing table, the peer forwards the request queries to the peer with identifier closer to the data's identifier in each step. In theory, most DHT-based systems can guarantee that the requested data can be discovered in $O(\log N)$ hops, where N is the number of peers in the P2P network.

The hash function distributes the identifier of the data over the whole range of identifier space even if there are many files whose information used for the input of the hash function resembles, which can balance the load between peers. However, DHTs can exhibit an $\Theta(\log n)$ imbalance factor, that is, some peers have $\Theta(\log n)$ times as much load as the average peer [SMK+01]. In order to mitigate the imbalance of load, [GLS+04, GS05] presents a virtual server approach where one physical peer stores one or more identifiers in DHT. By allocating the multiple separated region of the identifier space to one physical peer, the virtual server approach distributes the load on storage and bandwidth over the entire P2P network. Other approaches focus on the movement of peers to arbitrary locations in the identifier space where they are needed to fairly share the identifier space between peers [KR04].

The structured P2P network has a disadvantage in stability. When the continuous arrival and departure of peer occur, much information need to be exchanged between peers to maintain the entries in the routing table, and the routing table cannot keep the latest state, which may mislead the queries [RGRK04]. In order to improve the stability, some schemes have been proposed so far: for example, [MBR03] attempts to improve the stability by decreasing the number of entries in the routing table while keeping the search performance within an acceptable level.

Unstructured P2P network

In the unstructured P2P network, the search queries of the requested data are forwarded to the adjacent peers in each step until the requested data is found. It achieves the better search performance for the well replicated data than the structured one while it cannot reliably find the rare data [LRS02, CRB+03].

A traditional query forwarding method used in Gnutella, namely flooding, each peer which receives the query forwards it to all adjacent peers. In this method, the number of queries grows dramatically, which may finally saturate the network bandwidth [LCC+02]. On the other hand, Random Walk, where each peer forwards a query to a randomly chosen adjacent peer at each step until the requested data is found [LCC+02, CMS04], markedly decrease the number of queries at a cost of the efficiency and responsiveness [LW03].

In order to improve the responsiveness of the random walk, some proposals focus on the selection of the next peer of the query. By considering the peer's characteristics, queries can be forwarded to an appropriate peer. For example, it is probable that many replicas of the data are placed at few peers connecting to a large number of adjacent peers (degree). By considering it, in High Degree Forwarding (HDF) proposed in [AMG03], each peer forwards the query to high-degree peers with high probability so as to improve the search performance.

On the other hand, [LCC+02, CS02] have clarified that the appropriate replication method helps the random walk to find the requested data in short hops, and the optimal search performance can be achieved by allocating replicas of the data in Square-Root replication manner described in 2.1.2. It can be achieved by a simple replication method, Path Replication, where the replicas of the requested data are placed on the search path the search query goes through.

Furthermore, other important issue of the query forwarding method and the replication method is an improvement in the performance in load-balancing between peers. As I mentioned above, many replicas are placed at high-degree peers, so that, the number of occurrences of reading from and writing to the storage (storage load) is concentrated on them. In order to achieve the load-balancing, there are some proposals. At the query forwarding method point of view, an existing method based the on random walk, Low Degree Forwarding (LDF) proposed in [AMG03], forwards the query to small degree peer with high probability so as to mitigate the load on the high-degree peer. On the other hand, the replication method, LAR, proposed in [GSBK04] moves the replicas in peers with high-load to peers with low-load by comparing the resource utilization between peers. This method has the same concept as my research. I focus on the replication method where each peer determines to create and store a replica based on only the local information of peers.

2.3.2 Scheduling Algorithm for Grid Computing

In the Grid computing, scheduling algorithms consider the problem of allocating a application task with a large amount of data which can be divisible to a large number of independent, equal-sized chunks [Rob03, BGMR96], and attempt to optimize how to utilizes the resources are utilized for transmitting and processing the application data in terms of the application turnaround time and the system throughput. In this section, with respect to each metric, I introduce existing scheduling algorithms.

Maximize System Throughput

One of important issues of the scheduling algorithms is to maximize the system throughput (total load or number of instructions executed per unit time-period) in steady-state mode. Here, steady-state indicates the interval where all computers with computation capability are running at the maximum speed for the application processing.

Beauount et al. focus on the environment where all application tasks are generated by single node which is a root of the tree-graph [BCF+02]. The root node decides which tasks to execute in it, and how many tasks to forward to each of its children. And then, each child performs this action recursively. Beaumont et al. have clarified that the best allocation in the steady-state is bandwidth centric, where if enough bandwidth is available to the node, all children are kept busy; if bandwidth is limited, then tasks are allocated only to nodes which have sufficient bandwidth.

Marchal et al. considers the system throughput in a realistic network model where several nodes generates the application tasks and allocates them each other [MYCR05]. The proposed algorithm aims not only to maximize total throughput of all applications but also to achieve the fair balance among applications.

Minimize Application Turnaround Time

Minimizing the application turnaround time of applications having a large amount of application data is key issue of the scheduling algorithm. In order to complete this issue, Beaumont et al. has extended the bandwidth-centric approach to minimize the application turnaround time [BLR03]. This algorithm considers the tree-graph where the root node allocates application tasks to other nodes like in [BCF+02]. However, it basically determines the way to utilize the resources in a way to to maximize the throughput instead of the application turnaround time, so that, the application turnaround time cannot be minimized.

Uniform Multi-Round algorithm proposed in [YC03b, YC02] adopts a multiple-round manner where the application data is divided into an arbitrary number of chunks with an arbitrary size, and processed in multiple rounds. The proposed algorithm decides how the application data is divided and when the root node transmits the data to other nodes in a way to minimize the application turnaround time. In addition, Yang et al have extended/modified UMR in order to adapt to the practical computing environment where performance prediction error of resource capacity occurs [YC03a].

The bandwidth-centric approach can easily adapt to the heterogeneous environment in terms of the communication/computation capacities of nodes, but cannot optimize the application turnaround time. The objective of my research is to maximize the performance of each application by efficiently utilizing resources gathered by the resource management architecture. In order to complete our objective, I extend/modify the existing algorithm UMR to optimizing the application turnaround time and adapting to the heterogeneous environment.

Chapter 3

Replication Methods for Load Balancing on Distributed Storages in P2P Networks

3.1 Introduction

Because of recent improvements in end-computers and networks, the construction of a Peer-to-Peer (P2P) system over the Internet is now feasible. In a P2P system, each host, called a "peer", has both a client and server function, and information is exchanged through a direct connection between them. The P2P system dose not need a centralized server and can achieve robust information service by distributing information providers over the P2P network. Through P2P technologies, a massive storage system can be built virtually from distributed storage systems by collecting unused storage resources on peers over the Internet [BDET00, RD01b]. One of the advantages of these distributed storage systems is their use as a contents distribution network (e.g., internet radio streaming).

As an inherent feature of P2P networks, peers are very likely to leave the network, so some mechanism is required to enable users to get necessary data even if some peers are not connected to the network. For this purpose, that is, in order to achieve a robust data storage system in P2P networks, there have been several proposals for allocating replicas of original data on multiple peers. These replicas would also allow fast access to user's requiring data. Furthermore, replication could mitigate or avoid the concentration of access to a specific peer storing very popular data.

For these reasons, replication strategies play an important role. The replication strategies proposed so far mainly focus on improving the performance of the search for required data [LCC+02, AMG03]: e.g., a performance measure of concern is the ratio of queries which can discover the required information and the hop count needed. On the other hand, each peer is supposed to partly and equally devote its processing capability and storage capacity to queries from shared P2P applications, and the load due to queries should be thus balanced among the peers.

Therefore, in this chapter, I pay particular attention to the load on storage systems, which is due to reading and writing data items from and to the peer. I propose two replication methods of balancing the load on storages distributed over the P2P network while limiting the degradation of the search performance within some acceptable level. It should be noted that replication methods consist of such decisions as how many replicas should be allocated and where to allocate them. Among the replication methods proposed so far, the *Path Replication* method is of interest in terms of its good search performance and ease of implementation [LCC+02]. Thus, my proposed method will be based on this method, but will improve upon it. Furthermore, the performance of load balancing can depend upon the query forwarding method, which is a way to forward a query to discover the specific peer storing the required original data or its replica. As a query forwarding method, the *k-walker random walk* will be employed because it can discover requested data with a relatively small number of queries compared with other methods [LCC+02]. In addition, the P2P network is characterized by the so-called power law, i.e., the number of neighbor peers of each peer (degree) follows the power law [BA99, AB00, ALPH01]. In the power-law network, the load on storage due to queries can be concentrated on few peers with a high degree [SGG02]. By considering these features, I investigate the performance of my proposed replication methods through computer simulations.

The rest of this chapter is organized as follows. In Section 3.2, the procedure of data exchange between peers and the existing replication strategies are described. Section 3.3

presents my proposed replication methods. Section 3.4 outlines the computer simulations and presents the performance metric of the replication methods. In Section 3.5, I investigate the performance of my proposed methods. Finally, the conclusion is presented in Section 3.6.

3.2 Peer-to-Peer Network

In this chapter, I focus on a pure P2P network such as Gnutella and Freenet [CSWH00], where a server managing a directory of peers and their data is not required. In this section, I show the procedure of data exchange between peers in that system. In addition, I present some existing query forwarding methods and replication methods. Furthermore, the major problems of these methods are also described.

3.2.1 Information Acquisition in the P2P System

As shown in Fig. 3.1, queries are exchanged between peers to discover the requested data, because there is no centralized directory. The sender peer records its address and the name of the requested data on the query. Furthermore, I assume that the peer receiving the query also records its address on it. When the query arrives at some peer having the requested data, the peer sends the data out to the sender's address recorded in the query along the reverse sequence of addresses of the intermediate peers. During the data transmission, each intermediate peer determines whether or not it replicates that data according to the replication method.

3.2.2 Query Forwarding Method

The query forwarding method indicates how to transmit queries for discovering the peers storing the requested data or its replicas. Some existing methods are described below [LCC+02].

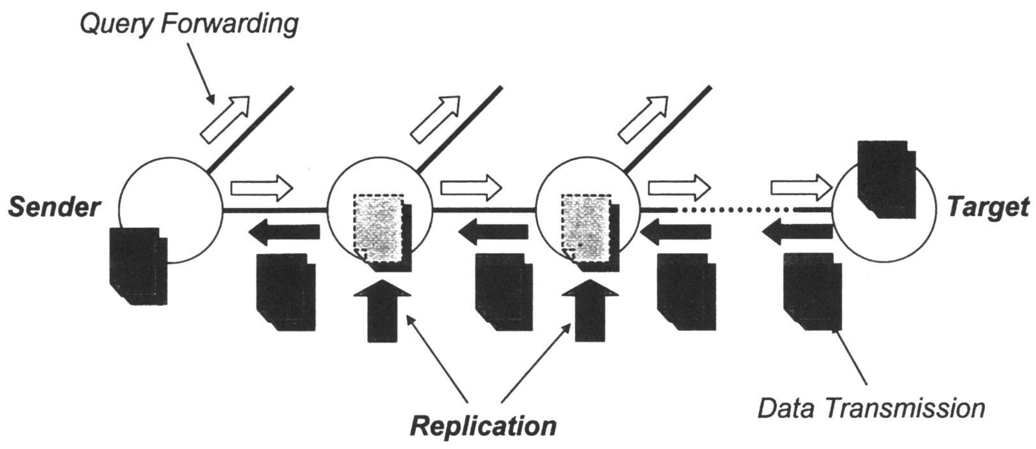

Figure 3.1: Procedure of the Data Exchange.

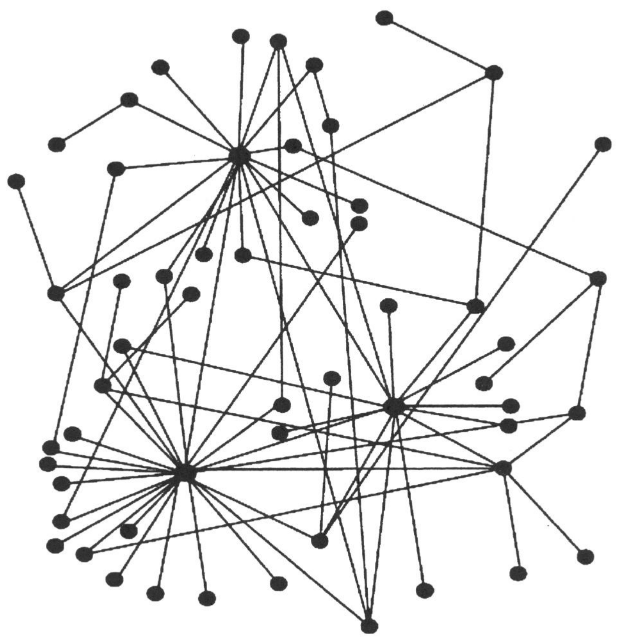

Figure 3.2: Power-law Network.

Flooding

The peer requesting the data forwards queries to all peers adjacent to it. Then, if the peers receiving the query do not have the requested data, they transmit queries to all their neighbors. This process is repeated until the requested data is discovered. Each query has a TTL (Time to Live) value, which limits the maximum number of intermediate peers through which the query can be forwarded. When the query is received by neighboring peers, the TTL value decreases by one, and when it becomes zero, the query is expired and deleted.

Expanding Ring

Peers first forward the query in the above *flooding* method with a small TTL value. If this query does not discover the requested data, the sender peer increases the TTL value and sends it out again. This process is repeated until the TTL value reaches some predetermined threshold.

k-walker Random Walk

The peer generates k queries and each of them is transmitted to a randomly chosen adjacent peer at each step. In this method, the TTL value is also set for each query. The TTL value sets the maximum number of hops.

Furthermore, *HDF (High Degree Forwarding)* and *LDF (Low Degree Forwarding)* are proposed in [AMG03]. In *HDF* and *LDF*, queries are forwarded to peers with a high degree and a low degree, respectively, by modifying the *k-walker random walk* based on the characteristics of the power-law network.

3.2.3 Replication Method

The replication method indicates a way to determine which peer is to store a replica, based on the P2P network status obtained in the query forwarding method. Some of the existing methods are described below [LCC+02, GSBK04].

Owner Replication [LCC+02]

Only the sender peer of the query stores a replica of the requested data.

Path Replication [LCC+02]

The requested data is replicated on all the peers along the data transmission path between the peer requesting the data and the peer having the data. This scheme has been employed in many distributed systems because of its good search performance and ease of implementation [RD01b, LCC+02, CSWH00].

LAR [GSBK04]

The sender peer of the search query stores the replica of the data stored in the peers receiving the search query. In addition, the peer is equipped with additional advanced functions. Each of the peers receiving the search query checks which is more heavily loaded among the sender peer and itself, and will allocate replicas of some of its own data if it is more congested in some sense (see [GSBK04] for further information).

Furthermore, in order to decrease the number of search hops, caches of the data are distributed over the P2P network. The cache contains the identifier and the location (i.e., IP address of the peer) of the data, so that, the cache enables the search query to be forwarded to the target peer only by IP routing, without any hop-by-hop forwarding over P2P network. In this method, the caches are distributed by being piggybacked onto the search queries.

3.2.4 Problems in the Existing Replication Methods

As shown in Fig. 3.2, the power-law network consists of a few peers with a high degree and a large number of peers with a low degree. In the case of *Path Replication*, the peer with a high degree forwards much more data than the peer with a low degree, so that a large number of replications will occur at the peers with a high degree. Therefore, the storage load due to writing and/or reading can be concentrated on a few high-degree peers, which thus play an important role in the P2P system. If the system fails due to overload or some other reason, a large amount of time is needed to recover the system [KLS02]. Therefore, we need a replication method to distribute the load over all the peers of the whole P2P network.

In *Owner Replication*, the number of replicas generated in the P2P network is limited to one at each data exchange, and so it takes a large amount of time to propagate replicas over the P2P network, thereby limiting the search performance for the requested data.

In addition, *LAR* needs much modification to the peers for providing additional functions mentioned above.

In my proposed method, each peer decides to create the replica based on only its local information, so that, the proposed method does not need much modification to the peers as described in Section 5.3, and can utilize the existing query forwarding methods without any modification.

3.3 Proposed Replication Methods

The *Path Replication* method explained above provides good performance in searching for the requested data. I thus focus on this method and will develop a way of load balancing for it. For this purpose, I propose the two methods below: *Path Random Replication* and *Path Adaptive Replication*. One is a rather straightforward extension of *Path Replication*. The other further improves upon it in its adaptability to the storage availability of the peers.

3.3.1 Path Random Replication

Path Replication places replicas in all the peers on the path the requested data goes to the requesting peer. The number of replicas created can become very large, which eventually may be more than necessary to achieve the required search performance. Thus, some amount

```
peer.initialize(){
    prob = replication ratio;
}
```

Figure 3.3: Initialization of Paramter in Path Random Replication

```
peer.decision(data){
    if(peer is sender peer){
        replication(data);
        return 0;
    }

    if(drand48() $\leq$ prob){
        replication(data);
    }
    return next.decision(data);
}
```

Figure 3.4: Decision of whether *peer* Replicates Requested Data or not in Path Random Replication

```
peer.replication(data){
    if(hold < capacity){
        storage_push(data);
    }
    else{
        storage_pop();
        storage_push(data);
    }
}
```

Figure 3.5: Replication of Requested Data in *peer*'s Storage

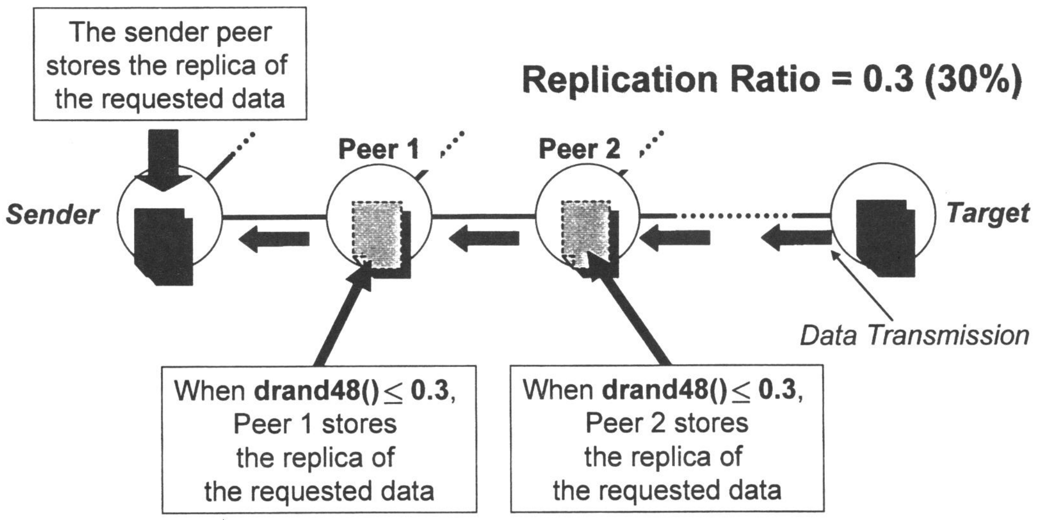
The sender peer stores the replica of the requested data
Replication Ratio = 0.3 (30%)
Peer 1
Peer 2
Sender
Target
Data Transmission
When drand48() ≤ 0.3, Peer 1 stores the replica of the requested data
When drand48() ≤ 0.3, Peer 2 stores the replica of the requested data

Figure 3.6: Sample Procedure of Path Random Replication

of the processing capability and storage capacity of the peers may be wasted, particularly on the few peers with a high degree. Nevertheless, there must exist an adequate number of replicas.

For this reason, I introduce the replication ratio, which is the ratio of the created replicas to all the intermediate peers on the path for each requested data. The replication ratio is determined in advance. Each intermediate peer randomly determines whether or not the replica is created and placed there, based on the probability of the pre-determined replication ratio. Thus, the *Path Replication* method coupled with this replication ratio is referred to as the *Path Random Replication* method, in which the special cases of replication ratios of 100% and 0% become the *Path Replication* and *Owner Replication* method, respectively. My major concern is determining an adequate replication ratio that will mitigate the concentration of load on the few high-degree peers while achieving almost the same search performance as *Path Replication.*

Figures 3.3–3.5 present the algorithm of *Path Random Replication* in pseudo-code. First, Fig. 3.3 indicates an initialization procedure when the peer *peer* invokes P2P application. As shown in this figure, on line 2, the replication probability *prob* of *peer* is set to the predetermined replication ratio, which is identical among peers in the P2P network. Next, Fig. 3.4 describes the procedure executed upon the requested data's arrival at *peer.* If *peer* is a sender peer of the search query, it stores the requested data in its own storage and treats the data as a replica as shown on line 2–5. Otherwise, it decides whether or not to replicate the requested data depending on its replication probability *prob* as shown on line 7–9, where **drand48()** produces a random number between 0 and 1. Finally, *peer* transmits the requested data to a next peer on the data transmission path on line 10. Here, the data placement procedure is described in Fig. 3.5. As shown in this figure, on line 2–4, if the number of replicas in its storage is smaller than the storage capacity, the replica of the requested data is placed on the storage. On the other hand, if there is no available storage capacity, *peer* stores the replica after removing the replica stored at the earliest time in the storage on line 5–8.

Figure 3.6 shows a sample procedure of *Path Random Replication* for the replication ratio of 0.3 (30%). When the query arrives at the target peer storing the requested data, the peer transmits the data to the sender peer of the search query in a hop-by-hop manner. In an example of Fig. 3.6, Peers 1 and 2 receive the requested data, and decide whether to replicate the data or not. Here, each of the peers obtains a random number between 0 and 1, and compares the number with the replication probability *prob* of 0.3 (Note that the replication probability *prob* of each peer is identical among peers in *Path Random Replication*). If the number is smaller than or equal to *prob*, the data is replicated in the storage of the peer. Finally, the sender peer receives the requested data and stores it.

3.3.2 Path Adaptive Replication

In *Path Random Replication*, the requested data is replicated in each intermediate peer on the path with a specified probability, which is the same at any peer in the P2P network. However, peers are highly different in their degree in the power-law networks. Consequently, the constant replication probability in *Path Random Replication* may still cause much load imbalance, because high-degree peers are frequently located in the data transmission path. Therefore, each peer should adaptively determine whether or not to create a replica depending on its resource status (e.g., available storage capacity). Thus, I propose *Path Adaptive Replication*, which determines the probability of the replication in each peer according to the predetermined replication ratio and its resource status.

In this research, the probability is defined in each peer as a function of x, which indicates its storage utilization ($0 \leq x \leq 1$); this allows each peer to determine whether or not to create a replica based on its own local information, without any global information over the P2P network.

Because of the nature of the power-law network, high-degree peers are often located on the data transmission path. Their storage capacity can be consumed substantially or sometimes exhaustively, while the storage capacity of the low-degree peers will not. Thus, in

```
$peer$.initialize(){
    $\lambda$ = init_lambda(replication ratio);
}
```

Figure 3.7: Initialization of Paramter in Path Adaptive Replication

```
$peer$.decision($data$){
    if($peer$ is sender peer){
        replication($data$);
        return 0;
    }

    $util = hold/capacity$;
    $prob$ = get_prob($util$);
    if(drand48() $\leq prob$){
        replication($data$);
    }
    return $next$.decision($data$);
}
```

Figure 3.8: Decision of whether *peer* Replicates Requested Data or not in Path Adaptive Replication

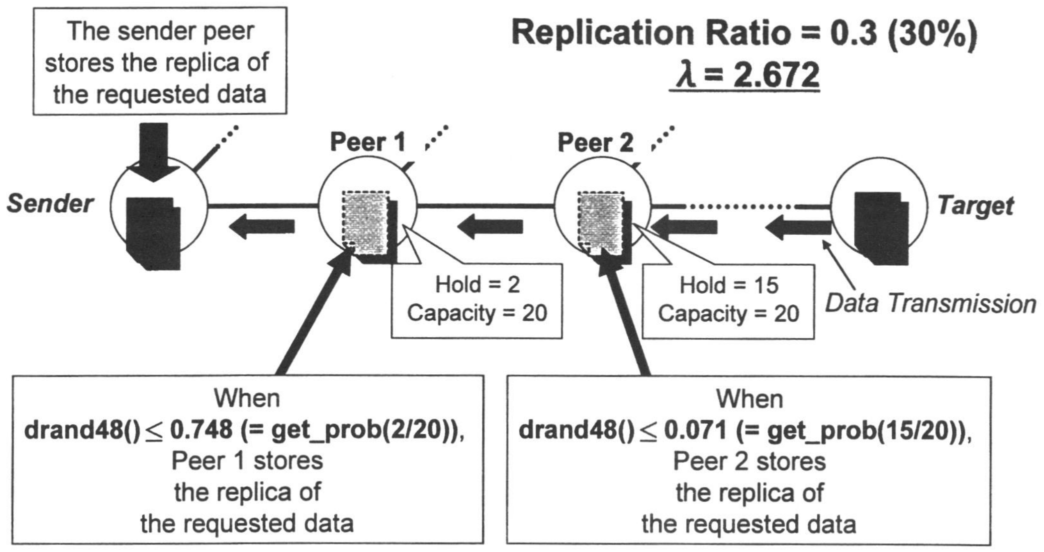
The sender peer stores the replica of the requested data
Replication Ratio = 0.3 (30%)
λ = 2.672
Peer 1
Peer 2
Sender
Target
Hold = 2
Capacity = 20
Hold = 15
Capacity = 20
Data Transmission
When drand48() ≤ 0.748 (= get_prob(2/20)), Peer 1 stores the replica of the requested data
When drand48() ≤ 0.071 (= get_prob(15/20)), Peer 2 stores the replica of the requested data

Figure 3.9: Sample Procedure of Path Adaptive Replication

order to distribute the load on the storage of the peers effectively, the replication probability, denoted by $f(x)$, in each peer should be monotonically decreased with the utilization x. Moreover, the high utilization of some peers may be because they frequently receive requested data. Thus, it is desirable that $f(x)$ decreases sharply with x in order to limit the total number of replications on peers which frequently receive data. Here, I introduce an exponential function $F(x) = 1 - e^{-\lambda x}$, and $f(x)$ must take a value from 0 to 1; so, I employ the following $f(x)$ as one candidate.

$$f(x) = 1 - \frac{F(x)}{F(1)} = 1 - \frac{1 - e^{-\lambda x}}{1 - e^{-\lambda}}. \tag{3.1}$$

In addition, I define the predetermined replication ratio $Ratio (0 \leq Ratio \leq 1)$, and λ in Eq. (3.1) is determined for the $Ratio$ as follows.

$$\int_0^1 f(x)dx = Ratio. \tag{3.2}$$

Figures 3.7 and 3.8 present the algorithm of *Path Adaptive Replication* in a pseudocode, except for the data placement procedure which is the same as that for *Path Random*

Replication. Figure 3.7 indicates an initialization procedure of the peer *peer*. As shown in this figure, on line 2, λ in Eq. (3.1) is determined in advance, where **init_lambda** derives λ corresponding to the predetermined replication ratio so as to satisfy Eq. (3.2). Furthermore, Fig. 3.8 describes the procedure executed upon the requested data's arrival at *peer*. The same as the procedure for *Path Random Replication*, the sender peer of the search query stores the requested data in its storage as shown on line 2–5. Otherwise, *peer* obtains its storage utilization *util* based on the number *hold* of replicas stored in the storage and the storage capacity *capacity*, and derives the replication probability *prob* on line 7, 8, where **get_prob** gives *prob* based on λ and *util* according to Eq. (3.1). Then, it decides whether to replicate the requested data or not by comparing the random number and *prob* as shown on line 9–11. Finally, *peer* transmits the requested data to a next peer on the data transmission path on line 12.

Figure 3.9 shows a sample procedure of *Path Adaptive Replication* for the replication ratio of 0.3 (30%), where each peer calculates $\lambda = 2.672$ based on the replication ratio according to Eq. (3.2) in advance. The same as the procedure for *Path Random Replication*, the target peer transmits the requested data to the sender peer in a hop-by-hop manner, and the sender peer stores the requested data in its storage. However, unlike in *Path Random Replication*, each peer derives the replication probability based on its own storage utilization as described above. In the example of Fig. 3.9, when Peers 1 and 2 receive the requested data, they set the replication probability *prob* to 0.748 and 0.071 based on the storage utilization *util* of 0.1 (= 2/20) and 0.75 (= 15/20) by Eq. (3.1), respectively. After that, each peer decides whether to store the replica by comparing the random number with *prob*. Therefore, Peer 1 stores the requested data in its storage with much higher probability than Peer 2.

3.4 Simulation Model

In this research, I investigate the performance of the proposed methods through computer simulations. I focus on the popular P2P file sharing system, namely Gnutella, where the degree of the peers follows the power-law distribution and about $10,000$ peers always organize its system [LCC+02, SGG02], and assume the Gnutella network as the simulation topology. Therefore, I employ the power-law network, generated by the topology generator shown in [BT02]. This power-law contains $10,000$ peers and $20,000$ links. Figure 3.10 shows the relationship between the degree of each peer and the number of corresponding peers. In addition, there are n kinds of data, and each of them is stored in 10 peers on the simulation topology; that is, $10 \times n$ data items are allocated in the P2P network in the initial condition.

I adopt the *k-walker random walk* as the query forwarding method, where k queries are randomly sent out from the sender peer to discover the requested data. In [LCC+02], the *k-walker random walk* with $k = 16, \cdots, 64$ queries results in good search performance in the power-law network which contains about $10,000$ peers. In this study, I focus on the replication methods, so that, I fix the number k of queries to 16 without investigating the impact of k. In addition, I consider cases where all queries reach their requested data so as to focus on the performance of the replication methods. In [LCC+02], the *k-walker random walk* discovers the requested data within a few more than 10 hops. Thus, I set TTL to 100, which is enough for each query to get the data in hops less than the TTL. Besides, each peer has the same storage capacity of c, which can accommodate c data. Furthermore, two or more of the same data are not stored in each peer, and when the replication occurs on a peer with no available storage capacity, the data or the replica stored at the earliest time will be eliminated from the storage in a FIFO (first-in first-out) manner. Therefore, each peer can store a maximum of c kinds of data.

In this chapter, I pay attention to the balance of the load on the storages distributed over the P2P network while limiting the degradation of the search performance within some acceptable level. First, as a measure of the search performance, I use the number of hops

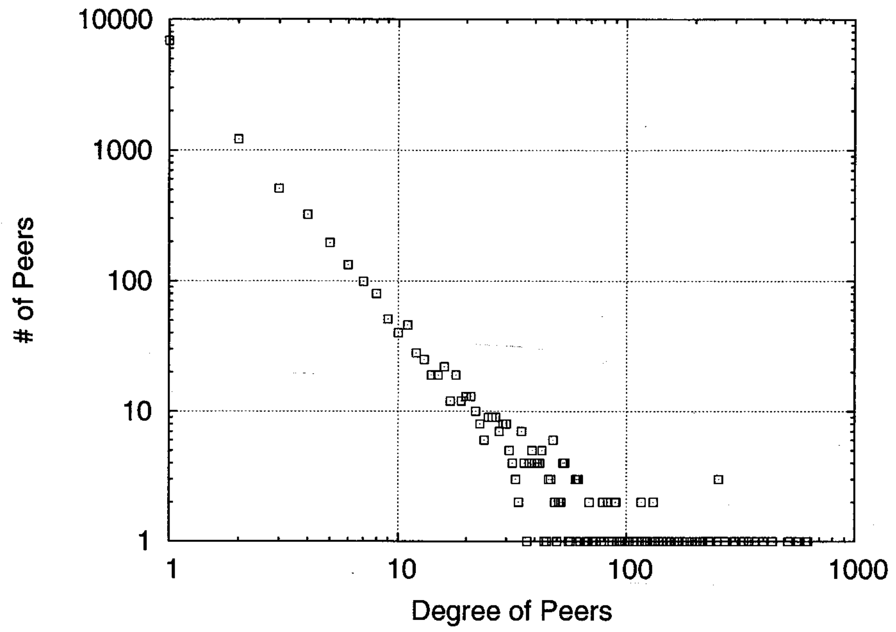

Figure 3.10: Degree in the P2P Network.

needed to discover the required data. In my proposed methods where each peer decides whether to create the replica of the data or not based on the replication ratio and its resource status, the number of replicas in the P2P network is smaller than that of *Path Replication*, so that the search performance may deteriorate to some extent. Therefore, I assume that users can accept an increase in the number of search hops up to, say, 120% of that of *Path Replication*. If the average hop count in the replication method with some replication ratio exceeds the acceptable level, I consider that it cannot achieve my objective, and then I do not adopt the associated simulation results even if it presents good load balancing.

Furthermore, existing research focuses on the number of queries arriving at each peer as a storage load [RD01b, GSBK04]. However, this represents the storage load due to reading only, not to both reading from and writing to the peer. Therefore, in order to represent the storage load appropriately, I define both reading and writing on each peer as a load, and the total number of reading and writing occurrences is considered as the load on the storage. In the performance evaluation results below, I show a figure that represents the storage load of the peers as a function of the peer's degree, and define the slope of the least-squares regression line as a metric of the load balancing, which will be referred to as the *load balance index*. A smaller index means better load balancing; in other words, the concentration of the load on the peers with a high degree is mitigated.

3.5 Simulation Results and Discussion

In this section, I show the performance of the proposed replication methods by focusing on the search performance and the balance of the storage load due to reading and writing of the requested data. First, I investigate the performance under the condition that no new data is added to the network during the simulation. Next, I consider the case where new data is added. I focus on the *load balance index* of $50,000$ queries and the average number of hops in the steady state, namely of $20,000$ queries after $10,000$ queries are issued. Furthermore, the peer which generates a query is selected according to the uniform distribution.

3.5.1 Comparison of the Search Performance of My Proposed Methods and That of Existing Methods

I show the effect of the replication ratio on the search performance in Fig. 3.11, which shows the average number of hops needed to discover the requested data as a function of the replication ratio for each method when the storage capacity c is set to 20, and the number n of kinds of data is set to 100. The threshold in the figure is 1.2 times as large as the average number of search hops of *Path Replication*; i.e., an increase of 20% is acceptable, as described in Section 3.4. As shown in this figure, the average number of hops exceeds the threshold for the low replication ratio, that is, for the replication ratio of less than 7% in *Path Random Replication* and less than 14% in *Path Adaptive Replication.* This is because in this range of small replication ratios, a sufficient number of replicas cannot be allocated in the P2P network. On the other hand, my proposed methods can maintain good search performance

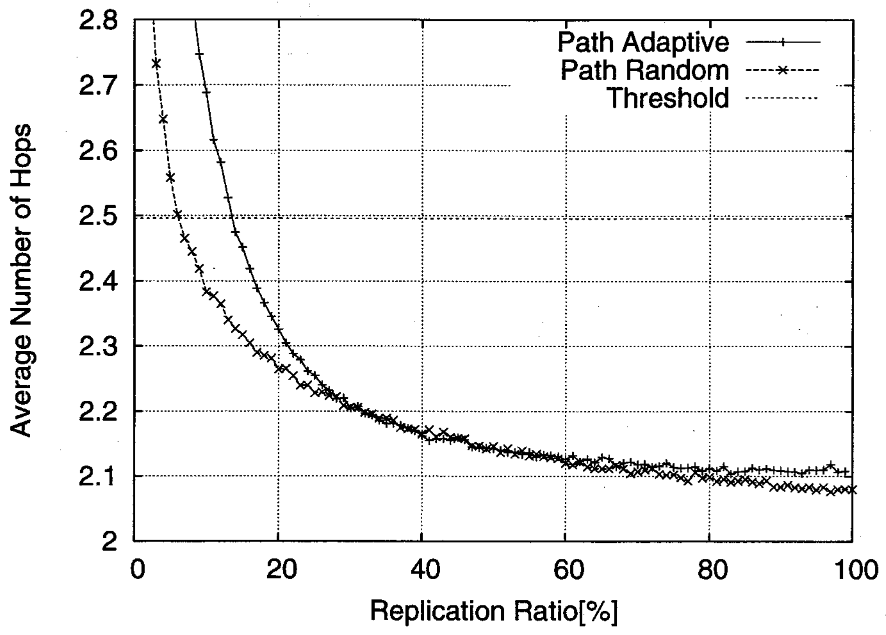

Figure 3.11: Number of Hops vs. Replication Ratio.

for a wide range of replication ratio.

In Fig. 3.11, we can see that *Path Random Replication* outperforms *Path Adaptive Replication* in the average number of search hops, but the difference is only less than 0.05. In addition, *Path Random Replication* achieves almost the same performance as *Path Replication* within the range of high replication ratios which are larger than 50%. This is because the proposed methods can allocate enough data items to high-degree peers to achieve good search performance, even if the replication ratio is lower than 100%.

3.5.2 Balance of Storage Load due to Reading and Writing Requested Data

Next I present the effect of varying replication ratios on the storage load due to reading and writing requested data items in *Path Random Replication* and *Path Adaptive Replication* when the storage capacity c is set to 20, and the number n of kinds of data is set to 100.

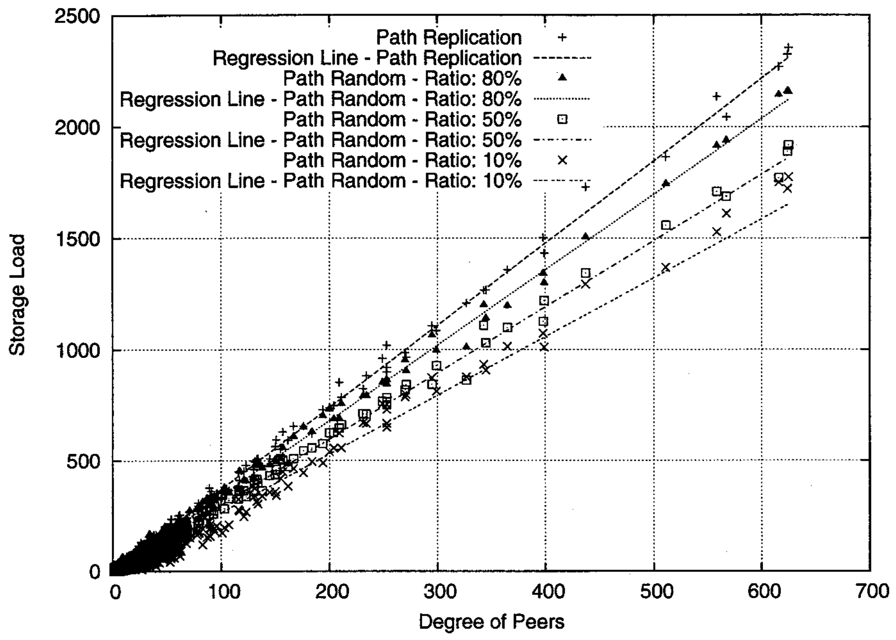

Figure 3.12: Storage Load vs. Degree of Peers (Path Random).

Figure 3.12 shows the relationship between the degree of the peers and the number of accesses to the storage of the corresponding peer in *Path Random Replication.* As shown in this figure, the number of accesses to the storage system increases with the replication ratio, and comes close to that of *Path Replication.* The load balance index described in Section 3.4 becomes larger as the replication ratio increases. The access is due to both writing and reading data, and the number of accesses due to each examined in Figs. 3.13 and 3.14. In *Path Random Replication,* the replication probability is the same at any peer and at any condition, and the high-degree peers are very likely to exist on the data transmission path. Therefore, as the replication ratio increases, the high-degree peers have a larger number of chances to write replicas, as shown in Fig. 3.13. On the contrary, the number of reading occurrences is inversely proportional to the replication ratio for peers with a degree larger than, say, 200, as shown in Fig. 3.14. The reason is as follows. In the case of a low replication ratio, the replicas are allocated to only the high-degree peers, which have many chances to be located on the data transmission path and they are very likely to have almost all the requested data

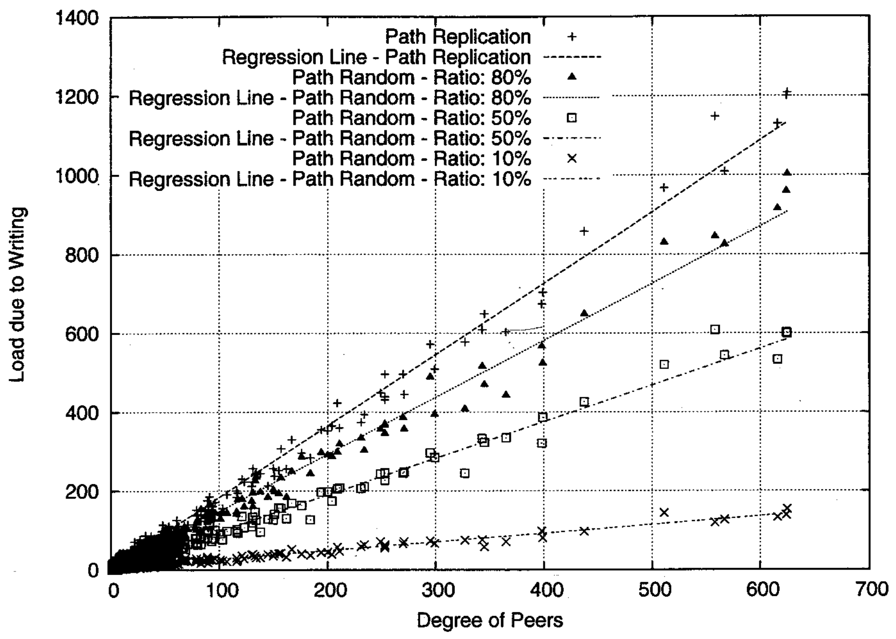

Figure 3.13: Load due to Writing vs. Degree of Peers (Path Random).

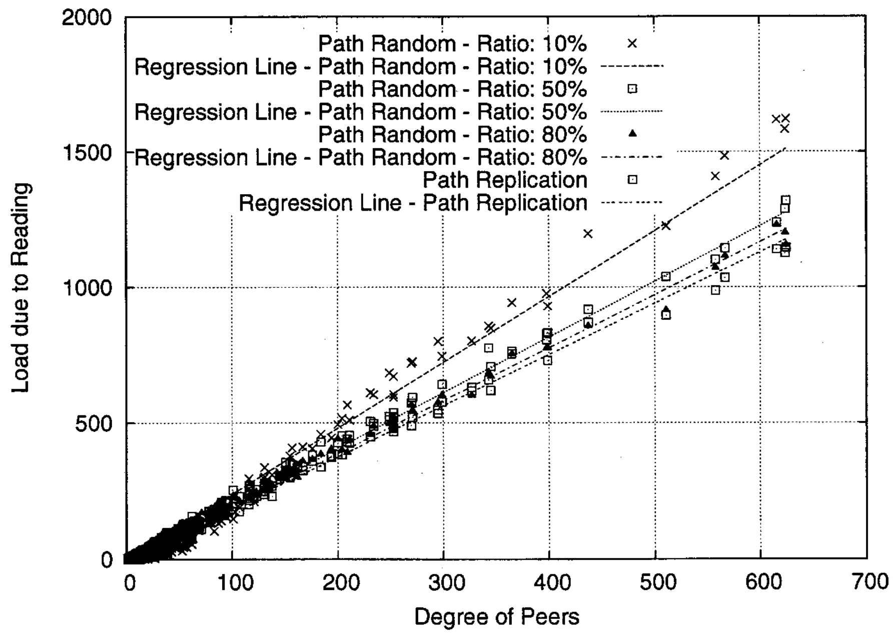

Figure 3.14: Load due to Reading vs. Degree of Peers (Path Random).

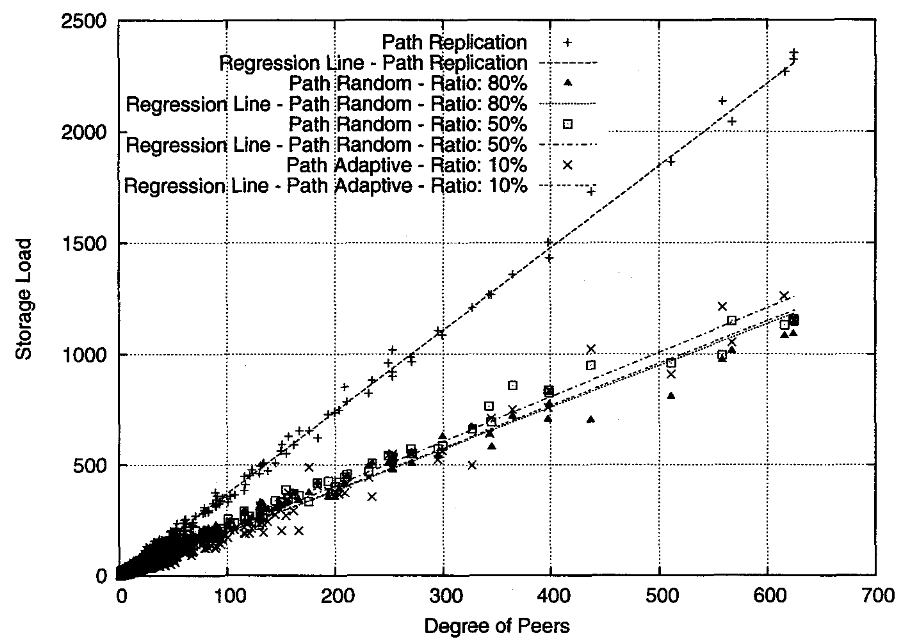

Figure 3.15: Storage Load vs. Degree of Peers (Path Adaptive).

items. With an increase in the replication ratio, the low-degree peers also have some replicas, so that the load due to reading is not concentrated on the high-degree peers. As shown in these figures, the characteristic of the number of writing occurrences is contrary to that of the number of reading occurrences. The total number of accesses due to writing and reading has a feature similar to the number of writing occurrences because the number of reading occurrences represents almost the same value for any replication ratio. As a result, the load balance index increases with the replication ratio.

Figure 3.15 shows the relationship between the degree of the peer and the number of storage accesses of the corresponding peer in *Path Adaptive Replication.* From this figure, we can see that the number of storage accesses of each peer is the same for any replication ratio, unlike that in *Path Random Replication.* In addition, the number of accesses is smaller than that of *Path Random Replication. Path Adaptive Replication* does not allocate replicas in a peer with a small available storage capacity. Therefore, the replicas are fairly distributed over all the peers, independently of their degree and replication ratio, as shown in Fig. 3.16.

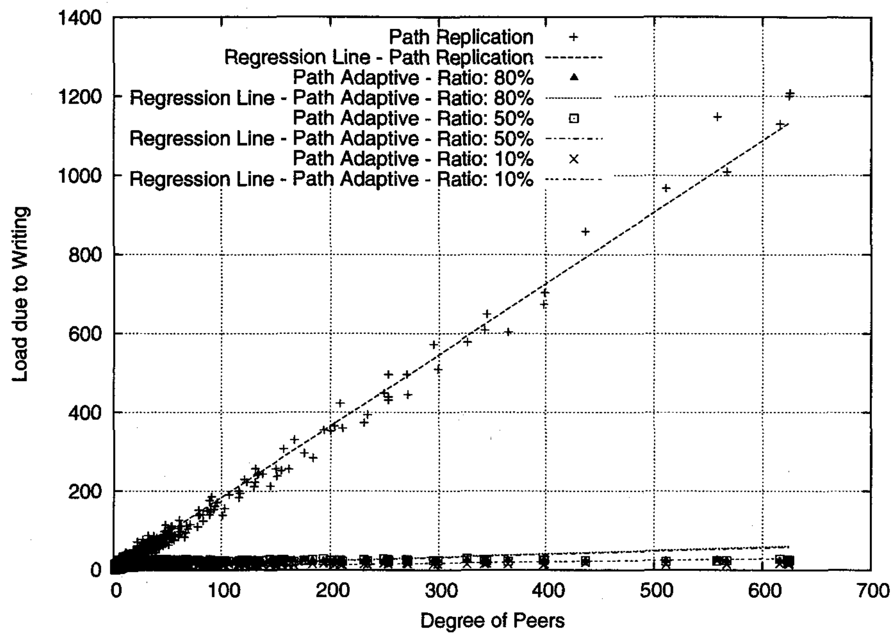

Figure 3.16: Load due to Writing vs. Degree of Peers (Path Adaptive).

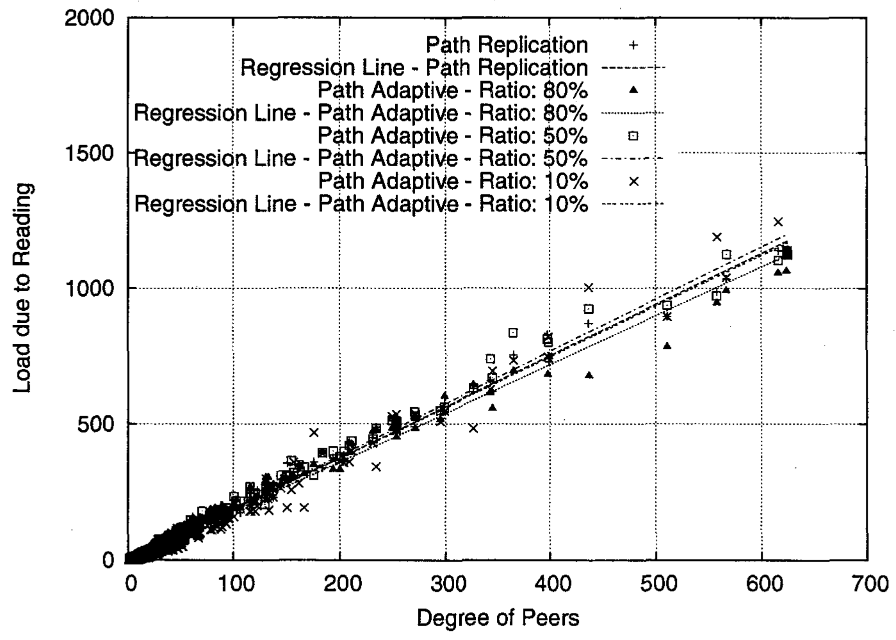

Figure 3.17: Load due to Reading vs. Degree of Peers (Path Adaptive).

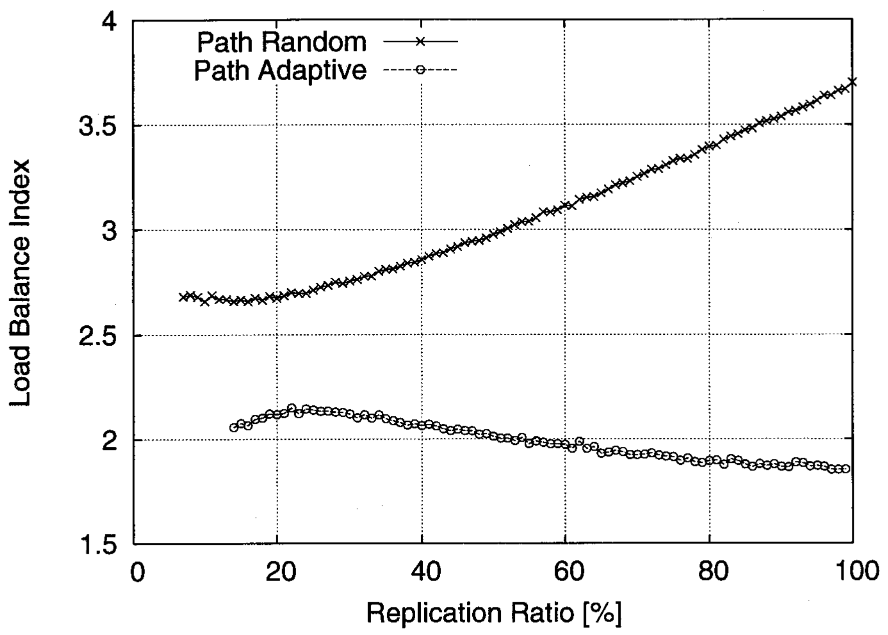

Figure 3.18: Load Balance Index vs. Replication Ratio.

On the other hand, as in *Path Replication* and *Path Random Replication*, queries in *Path Adaptive Replication* can more frequently discover the requested data in peers with a higher degree, as shown in Fig. 3.17. Since the number of writing occurrences is almost the same for any replication ratio, the total number of accesses is almost the same as the number of reading occurrences, as shown in Fig. 3.15.

I present the load balance index as a function of the replication ratio in Fig. 3.18. As described in Section 3.4, the load balance index indicates how evenly the load is distributed. As shown in this figure, the index of *Path Adaptive Replication* is smaller than that of *Path Random Replication* for any replication ratio, and *Path Adaptive Replication* improves its performance in load-balancing as the replication ratio increases. This is because, as the replication ratio increases, the storage load in *Path Random Replication* increases, and the storage load in *Path Adaptive Replication* decreases, as described previously. Thus, *Path Adaptive Replication* with a high replication ratio achieves excellent performance in load balancing.

3.5.3 Impact of Simulation Parameters on Performance of Proposed Methods

In this subsection, in order to investigate the impact of some simulation parameters, namely the storage capacity c and the number n of kinds of data, I examine the search performance and the performance in load-balancing of the proposed replication methods when c is set to 40 and n is set to $1,000$, while c and n are set to 20 and 100 respectively, until now.

Figure 3.19 shows the average number of hops needed to discover the requested data as a function of the replication ratio. As shown in this figure, both of *Path Random Replication* and *Path Adaptive Replication* achieve the number of hops within 120% of that of *Path Replication* for a wide range of replication ratio, like in Fig. 3.11 presented in Section 3.5.1.

Furthermore, Fig. 3.20 shows the load balance index as a function of the replication ratio. In this figure, as in Fig. 3.18 of Section 3.5.2, the index of *Path Adaptive Replication* is smaller than that of *Path Random Replication*, and *Path Adaptive Replication* improves its performance in load-balancing as the replication ratio increases.

Therefore, we can see the improvement in load balancing by *Path Random Replication* and *Path Adaptive Replication* in this case as well, while the load balance index takes different values for different c and n.

3.5.4 Path Adaptive Replication with Priority Level

So far, I have considered the case where no new data is added to the network during the simulation. In actual networks, new data is added in the P2P system at run time, so that the replication method should be adaptive to such situations. As stated previously, *Path Adaptive Replication* performs very well, but it does not allocate any replicas on the peers with no available storage capacity except for the requesting peers which can store its requesting data in its own storage, so that the new data cannot be replicated after the storage is occupied by old data. This leads to critical degradation of the search performance for the new data.

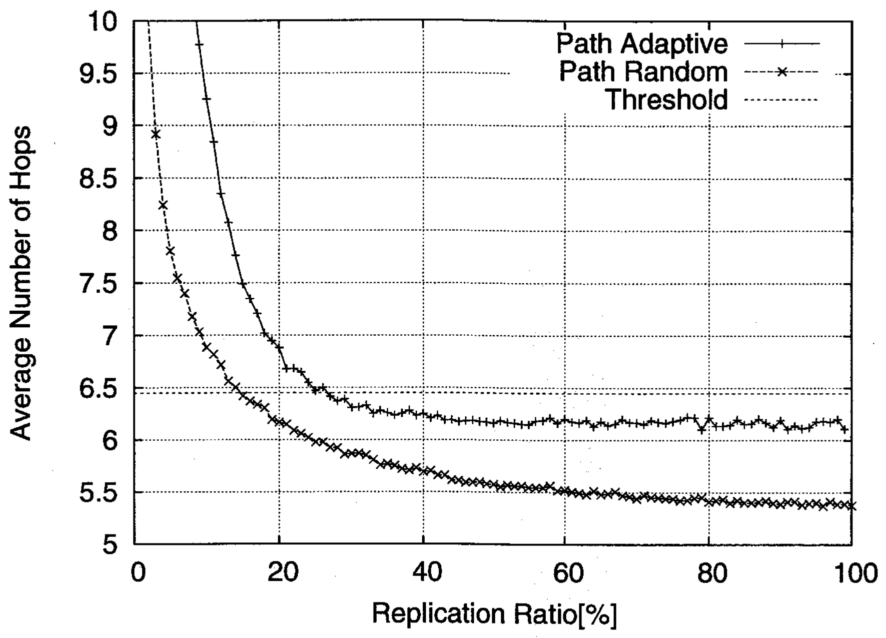

Figure 3.19: Number of Hops vs. Replication Ratio (n = 1000, c = 40).

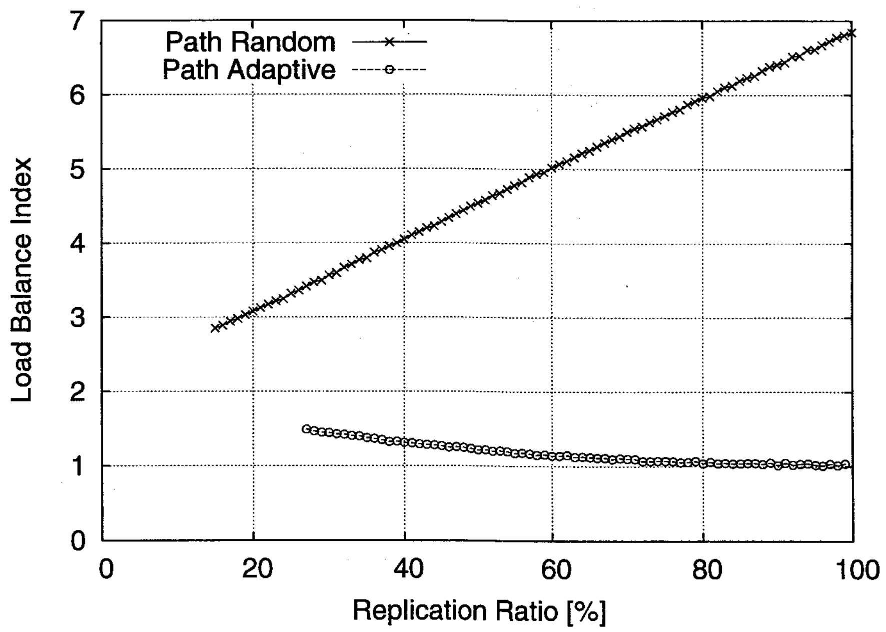

Figure 3.20: Load Balance Index vs. Replication Ratio (n = 1000, c = 40).

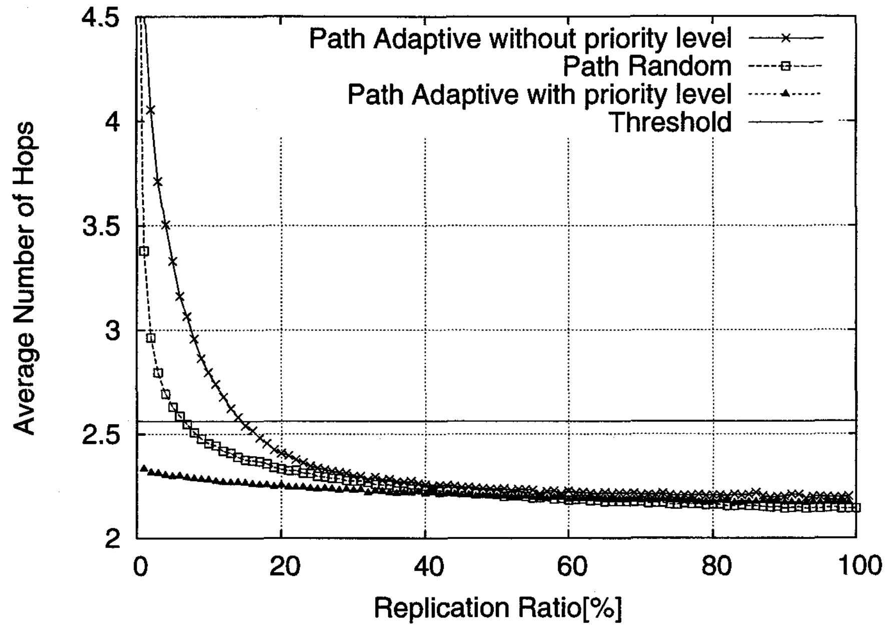

Figure 3.21: Number of Hops vs. Replication Ratio.

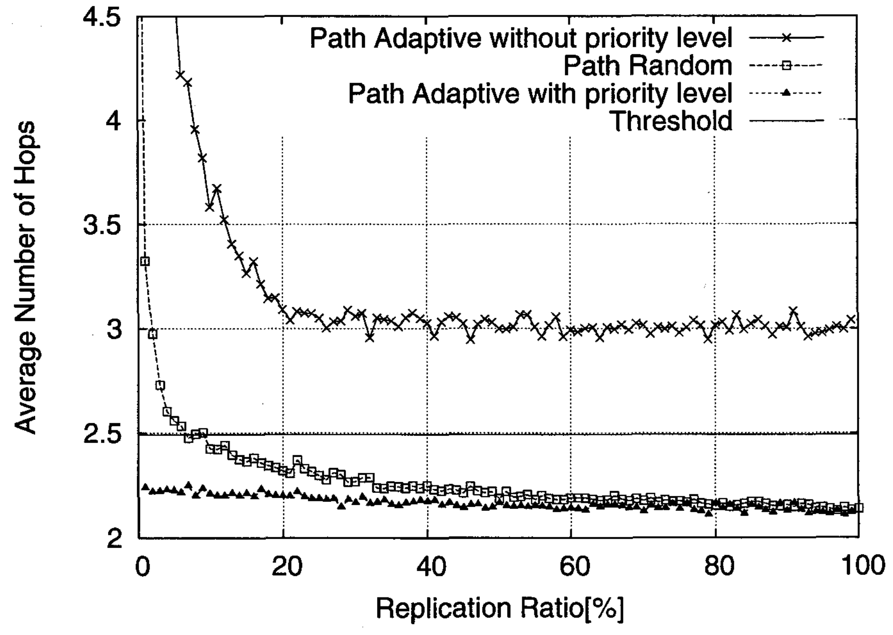

Figure 3.22: Number of Hops vs. Repliation Ratio (Added Data Only).

Therefore, I modify *Path Adaptive Replication* to allow new data to be distributed over the peers. In this subsection, x in Eqs. (3.1) and (3.2) is redefined as follows.

$$x = \frac{Storage\ utilization\ +\ Priority\ level}{2}. \tag{3.3}$$

The *Priority level*, denoted by δ, is a parameter used to adjust the replication probability in order to allow newer data to be replicated more frequently. The *Priority level* parameter is associated with how much new data is requested in the peer storing it. More specifically, I let m be the total number of data stored in the peer. And I let n indicate how much new requested data is among all the data there, and n is set at 1 for the newest data. δ is given by n/m. This allows new data to have priority over old data.

In this evaluation, ten kinds of data are added to the P2P network and each kind is allocated to ten peers after $10,000$ queries are issued. Then, I investigate the performance of the proposed method. For the newly added data, I also focus on the average number of hops in the steady state, namely of $20,000$ queries after $20,000$ queries are issued. Figures 3.21 and 3.22 show the number of hops needed to discover the requested data as a function of the replication ratio. From Fig. 3.21, *Path Random Replication*, *Path Adaptive Replication* without the priority level and that with the priority level attain almost the same search performance close to that of Path Replication for a wide range of replication ratios. On the other hand, as shown in Fig. 3.22, the average number of hops needed to discover the new data in *Path Adaptive Replication* without the priority level is equal to almost 3, while that of *Path Adaptive Replication* with the priority level stays at less than 2.2. Thus, it should be noted that, for newly added data, *Path Adaptive Replication* without a priority level cannot be accepted in terms of search performance.

Furthermore, Fig. 3.23 illustrates the load balance index of each peer as a function of the replication ratio for each of the replication methods. As shown in this figure, *Path Adaptive Replication* without the priority level achieves the most effective load balancing, but its search performance for new data is degraded compared with other methods, and the differ-

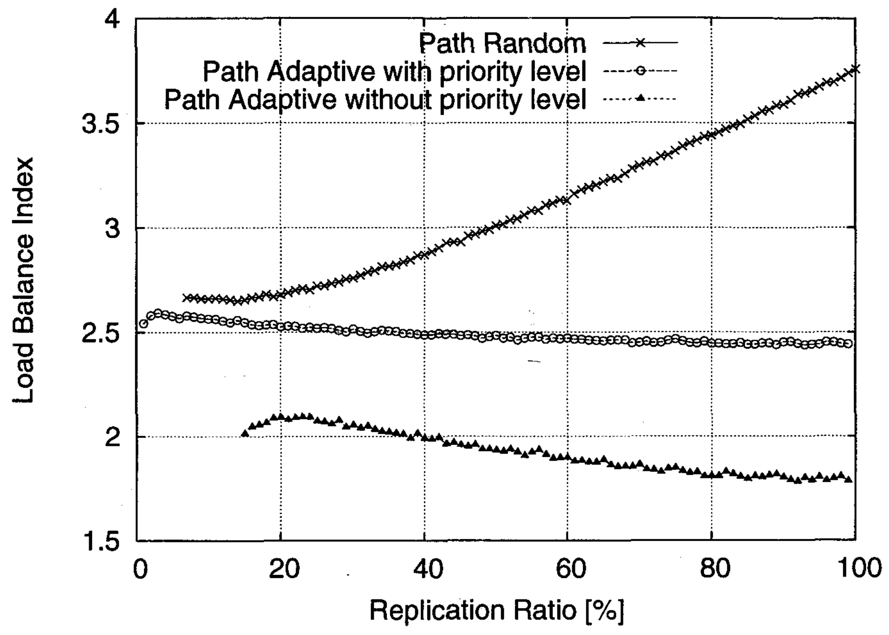

Figure 3.23: Load Balance Index vs. Replication Ratio.

ence of the average number of hops becomes more than 0.7. On the contrary, *Path Adaptive Replication* with the priority level can achieve excellent search performance even for new data, and is superior to *Path Random Replication* in terms of load balancing. Therefore, I can say that *Path Adaptive Replication* with the priority level can successfully distribute the load on storages over P2P networks while keeping the degradation of the search performance within an acceptable level.

3.6 Conclusion

In the P2P network, the number of neighbors (degree) of each peer follows the power law; there exist a few high-degree peers and a large number of low-degree peers. Therefore, a huge number of requests can go through these few high-degree peers, and the storage loads due to reading and writing requested data is concentrated on them. Thus, in order to mitigate the load concentration on the high-degree peers over the P2P network without deteriorating the search performance too much, I proposed two replication methods, which make replicas on some chosen peers, through which the data passes. One method, *Path Random Replication*, chooses the peers randomly with a predetermined replication ratio, and the another method, *Path Adaptive Replication*, further improves the procedure in the decision to make a replica on a peer depends on how much storage is still available on it as well as the predetermined replication ratio. I have evaluated the performances of both methods through computer simulations.

My results show that *Path Random Replication* could balance the loads on the storage of the peers very well at a cost of deterioration of the search performance of only 10% in comparison with that of the existing replication method (*Path Replication*). Also, *Path Adaptive Replication* could further improve the load balancing using each peer's local information on resource availability. Furthermore, I considered the case where new data is added to the system; in order for the system to work well, I modified *Path Adaptive Replication* by considering when the required data is allocated on the peer as well as considering the information on resource availability. *Path Adaptive Replication* achieved effective load balancing while limiting the degradation of the search performance within an acceptable level.

Chapter 4

Performance Comparison of Task Allocation Schemes Depending upon Resource Availability in a Grid Computing Environment

4.1 Introduction

Because of recent improvements in the performance of end-computers and networks, the construction of a Grid system over the Internet is now feasible. A Grid system [FK98a] is constructed by connecting geographically distributed computers over the Internet to secure greater computing power. Several fundamental services are indispensable for this sharing of computational resources (e.g., CPU time, memory, storage, and so on) in a Grid environment: security management, resource management, and task execution on remote hosts, all of which are run as low-level middleware [FKT01]. The Globus Toolkit [FK98b] produced by the Globus Alliance [Glo] is a well-known example of middleware providing these important services. Using these services, a computer (user) can securely submit a set of task executions to the Grid environment, while requiring a scheduling system linked to the middleware to appropriately select the computers to which the tasks will be allocated.

The Grid system is very likely to be heterogeneous; that is, each of its computers has a distinct computing performance. In order to achieve good performance of Grid computing in this environment, several scheduling algorithms have been proposed [YC03b, BLR03]. These algorithms can adapt themselves to a heterogeneous environment by considering available computing and networking resources in the Grid environment, and by dividing a long-lived task composing an application into an appropriate number of sub-tasks. The algorithms assume that the scheduler can always perceive all information about the task allocations as well as the latest resource status in the Grid environment, as in cluster computing.

However, the Grid system is actually built on a global network, where the computing and networking resources are shared by many organizations with their own objectives, and each organization can independently allocate their jobs on their resources by their own schedulers. Therefore, Grid computing resources are very likely to be used unevenly, and will not be absolutely managed in a centralized manner [FKT01]. Furthermore, each scheduler cannot always perceive the dynamic variations of the computing resource status, and can only refer to the information periodically updated through a directory service which manages the resource status in the Grid environment in a distributed manner (e.g., MDS (Metacomputing Directory Service) [CFFK01]). These Grid characteristics lead to some uncertainty in the information on the resource status: in particular, the obtained information may not indicate the current status exactly. Most schedulers which have been proposed so far don't focus on this effect on the performance of task allocation. Therefore, the task allocation scheme in the Grid environment should be designed with consideration of the fact that resources distributed over the Internet are used unevenly and the obtainable information on their utilization can involve some uncertainty.

In this chapter, I investigate the preliminary performance of some task allocation schemes by considering the unevenness of resource utilization and the uncertainty of utilization information. My investigation is based on a simple analysis and computer simulation, through which I focus on CPU performance and availability as a function of utilization. In the analysis, the computers are modeled as an M/G/1-PS (processor sharing) queue, and the average turnaround time of the application with a long-lived task in each scheme is derived analytically. Adopting the M/G/1-PS queue allows the average time for each computer to process a sub-task to be a function of CPU utilization alone. Since the Grid computing environment is constantly evolving, it is very difficult to precisely describe the CPU utilization of computers comprising the Grid environment. Hence, a probability distribution function associated with CPU utilization is introduced to describe the uneven utilization of the computers. In order to verify the accuracy of the analysis, I compare the performance derived by the analysis with that derived by the simulation. Then, I consider a more realistic simulation model in which the available information on resource utilization is uncertain due to the reasons mentioned previously, while the available information is assumed to exactly indicate the current status in the analytical model.

The rest of this chapter is organized as follows. In Section 4.2, the resource management model of Grid computing is described, and the task allocation schemes are defined. Section 4.3 outlines the analytical evaluation of the average application turnaround time, and examines the characteristics of the proposed task allocation schemes. In Section 4.4, I deal with a realistic computer simulation and investigate the impact of information uncertainty on the performance of each scheme. Finally, the conclusion is presented in Section 4.5.

4.2 Grid Computing Architecture

4.2.1 Resource Management Architecture

The resource management architecture employed here consists of computers and directories, as shown in Fig. 4.1.

Computer

Every computer is assumed to manage its own performance and capability using a database. This information is measured periodically using a resource management toolkit such as Net-

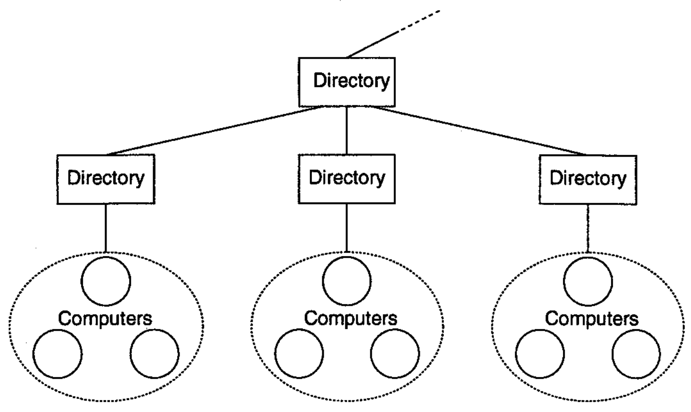

Figure 4.1: Resource Management Architecture.

work Weather Service [WSH99]. In this context, computers execute tasks allocated by other computers in the Grid, while simultaneously executing the necessary local tasks.

Here, CPU performance is denoted by S_i [instructions per unit time], and CPU utilization ρ_i ($0 \leq \rho_i < 1$) is defined as the load imposed on the computer R_i. The computer can execute tasks allocated to it at the rate of S_i, and ρ_i of the total capability of the CPU is consumed when the user allocates new sub-task.

Directory

The directory manages the static computer information as a database, including the operating system, processing capability, memory capacity, and IP addresses. The database is updated periodically when computers notify the directory. According to requests from users, the directory asks computers under its control to send back information related to the resource status, which is cached in the directory database, and then replies to users with the obtained information. Each entry registered with a database has a lifetime [GCKF01]; if the entry has not been updated during the lifetime, the directory determines that the relevant computer is no longer available.

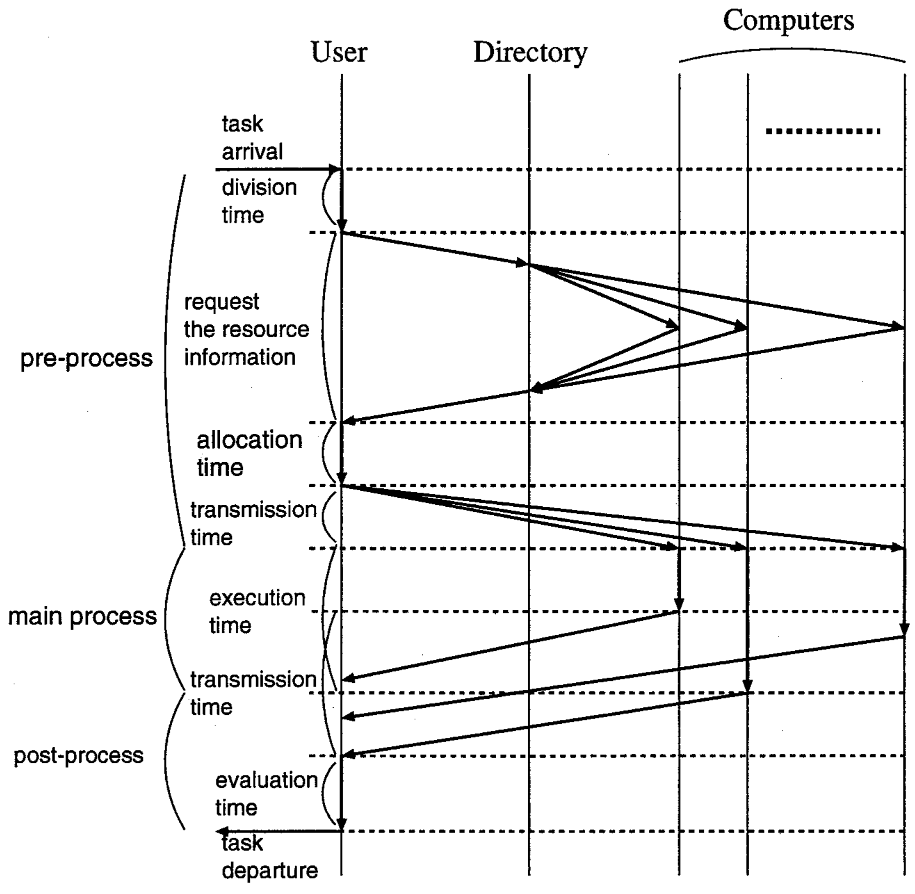

Figure 4.2: Grid Computing.

4.2.2 Grid Computing

The Grid computing process is outlined in Fig. 4.2. After a task arrives, the user divides long-lived tasks into n independent, small-scale sub-tasks. The long-lived task is composed of W instructions. Here, the overhead of each sub-task due to segmentation is neglected for simplicity, and so the number of instructions in each sub-task W_s can be defined as W/n. Next, the user requests the directory to investigate the resource status of the computers, and to reply with the relevant information. In this study, the limited case of a maximum of one sub-task allocated to each computer is considered, so that the user selects n computers to which sub-tasks are allocated based on the information from the directory. The computer resource information is presented in detail in Section 4.3. In this study, it is reasonably assumed that all tasks are independent, have the same priority, and are fairly scheduled in a round-robin manner on all computers. The overall application turnaround time is determined by the time when the result of the latest sub-task is returned.

As shown in Fig. 4.2, the interval from the allocation of sub-tasks to the completion of all sub-tasks is defined as the *main process*. Processes to be executed before the main process begins are defined as the *pre-process*, and those executed after the main process are defined as the *post-process*. In this chapter, I define the application turnaround time as the time required for the main process, and the performance of each sub-task allocation scheme is evaluated based on it to show the impact of only the CPU status on the performance.

4.2.3 Sub-task Allocation Scheme

I focus on the following three sub-task allocation schemes to investigate the impact of the uneven CPU utilization in computers on their performance. As I mentioned in Section 4.2.2, I consider the limited case of a maximum of one sub-task allocated to each computer.

Scheme 1 (Random Selection)

The user randomly selects n (number of sub-tasks) computers from all the available computers, then allocates all the n sub-tasks to the selected computers. In this scheme, the user does not have to consider the computer resource information, thereby minimizing the pre-process time.

Scheme 2 (Resource Availability First Selection)

The user selects n least-loaded computers and allocates sub-tasks to these computers. In this scheme, the user needs to receive the current resource information for all computers and compare the available processing capability. This results in a lengthy pre-process time.

Scheme 3 (Random Selection with Resource Availability Threshold)

The user requests the directory to ask all computers to send back a reply if the computer satisfies the requirement that the current utilization is lower than some threshold Th. The user then randomly selects n computers among those responding to the inquiry, and allocates sub-tasks accordingly. When Th is set relatively low, fewer than n computers may satisfy the requirement, in which case the user increases Th by some amount and re-investigates the resource status. This process is repeated until n or more computers satisfy the requirement. The pre-process time of Scheme 3 is expected to be shorter than that of Scheme 2 because the user does not need to receive resource information from all available computers, and also does not need to compare the information, unlike in Scheme 2. However, in this study, in order to investigate the preliminary performance of task allocation schemes, I assume that the pre-process time is small enough to be neglected compared with the application turnaround time mentioned in Section 4.2.2.

4.3 Analysis

4.3.1 Analytical Model

I employ the analytical model illustrated in Fig. 4.3, in which the user is directly connected to 120 computers over the Internet. In this section, I analytically show the impact of the uneven CPU utilization in computers on the performance of proposed task allocation schemes, in which I thus neglect the transmission delay between the user and computers. The performance of each scheme described in Section 4.2.3 is evaluated using this model. Computers are evenly divided into three groups in terms of CPU performance: classes A, B and C correspond to CPU performances S_A, S_B and S_C respectively. In what follows, I assume $S_A = 50$, $S_B = 100$ and $S_C = 150$. Here, I assume a basic CPU with a performance of 1.0, and define one unit period as the time required for the basic CPU to calculate the task of unit instruction. Thus a CPU with a performance of 50, which is fifty times faster than the basic CPU,

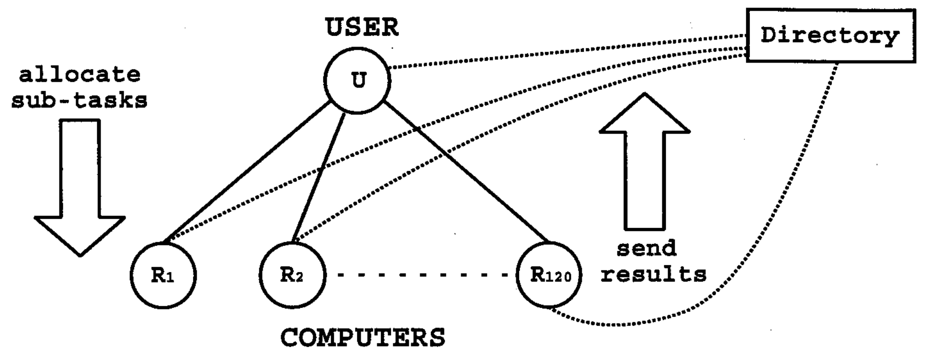

Figure 4.3: Analytical Model.

can calculate a task with a workload of 100 in 2 unit periods. Because only the main process is considered here, the average application turnaround time is adopted as a performance measure, and can be obtained analytically as follows.

Each computer is modeled by the M/G/1-PS queue, where all tasks in some computer are fairly served with a round-robin scheduler. Thus, this model allows the coexistence of other tasks, which may be allocated from other users and/or may be generated by the local computer. In this model, I assume that the workload W_s of sub-task injected into computer R_i by the user is small, and this does not change the long-term utilization ρ_i of the computer, that is, I focus on the Grid environment where the load due to tasks generated by a computer itself is dominant over the entire load of the computers. Thus the average time $E[T_i]$ required for the computer R_i with CPU performance S_i is given by

$$E[T_i] = \frac{W_s/S_i}{1-\rho_i}. \tag{4.1}$$

I consider that the application turnaround time is defined as the interval between the allocation and the completion of all sub-tasks [BGMR96]. If one task is divided into n sub-tasks, which are allocated to computers $R_1, \cdots, R_n$, the application turnaround time is defined as follows.

$$E[T] = E[\max\{T_1, \cdots, T_n\}]. \tag{4.2}$$

However, it is difficult to analytically derive the average application turnaround time by solving Eq. (4.2), so that, in this chapter, I assume the average application turnaround time as the maximum value of the sub-task processing time $E[T_i]$ for simplicity as Eq (4.3), and investigate the preliminary performance of task allocation schemes.

$$E[T] = \max\{E[T_1], \cdots, E[T_n]\}. \tag{4.3}$$

In fact, these equations have a magnitude relationship as follows.

$$E[\max\{T_1, \cdots, T_n\}] \geq \max\{E[T_1], \cdots, E[T_n]\}. \tag{4.4}$$

I will investigate the effect of it on the performance of task allocation schemes in Section 4.4 by comparing the analytical with the simulation results.

In this chapter, I define the current resource capability I_i of computer R_i as the average virtual turnaround time for a task with a unit workload would experience if it is allocated to the computer as follows.

$$I_i = \frac{1/S_i}{1-\rho_i}. \tag{4.5}$$

Even if the CPU performance of the three groups differs, the user can select computers in order of the value I_i to allocate sub-tasks in Scheme 2 (Resource Availability First Selection). In Scheme 3 (Random Selection with Resource Availability Threshold), the user randomly selects computers with current resource capability I_i lower than the predetermined threshold Th. Thus, each scheme mentioned in Section 4.2.3 is adaptable to the analytical model, which models computers with different CPU performances, and is evaluated as follows:

1. Computer resource information is investigated using Eq. (4.5).
2. Sub-tasks are allocated to n computers according to their current resource capability.
3. The average application turnaround time is evaluated using Eq. (4.3).

However, in general, the CPU utilization changes dynamically. Therefore, 10,000 experiments were conducted under the condition that the CPU utilization of each computer is changed at every trial while the average CPU utilization of available computers $\bar{\rho}$ is set to a constant value ($0 \leq \bar{\rho} < 1$). The mean of the average application turnaround time for all experiments is then employed as the performance measure. The Grid computing environment is constantly evolving, which makes it very difficult to precisely describe the CPU utilization of the computers involved. Thus, I introduce a power distribution as the CPU utilization distribution because it appears in many phenomena in the distributed environment (e.g., the number of accesses received by web servers [New05, AH99]), which may be suited for the unevenness of CPU utilization of computers. In [New05], the distribution of the CPU utilization ρ_i of computer R_i is formulated as follows.

$$\Pr\{\rho_i \leq x\} = x^a. \quad \left(i = 1, \cdots, 120, \quad a = \frac{\bar{\rho}}{1-\bar{\rho}}\right) \tag{4.6}$$

4.3.2 Analytical Results and Discussion

Average Application Turnaround time vs. Average CPU Utilization

The workload of each task W (i.e., the number of instructions) is set to 10,000 and the number of sub-tasks n is set to 40. Therefore, the workload of each sub-task W_s becomes 250 (=10000/40).

Figure 4.4 shows the average application turnaround time as a function of average CPU utilization. In Random Selection (Scheme 1), 40 computers are randomly chosen independent of CPU utilization or class. Therefore, some of the sub-tasks will be assigned to computers of class A, resulting in a turnaround time of as long as 5 (=250/50), even when the average CPU utilization is 0. With the power distribution, allocation under Scheme 1 may result in assignment to a highly loaded computer. As a result, the average application turnaround time increases markedly with average CPU utilization.

In Resource Availability First Selection (Scheme 2), the user selects n least-loaded com-

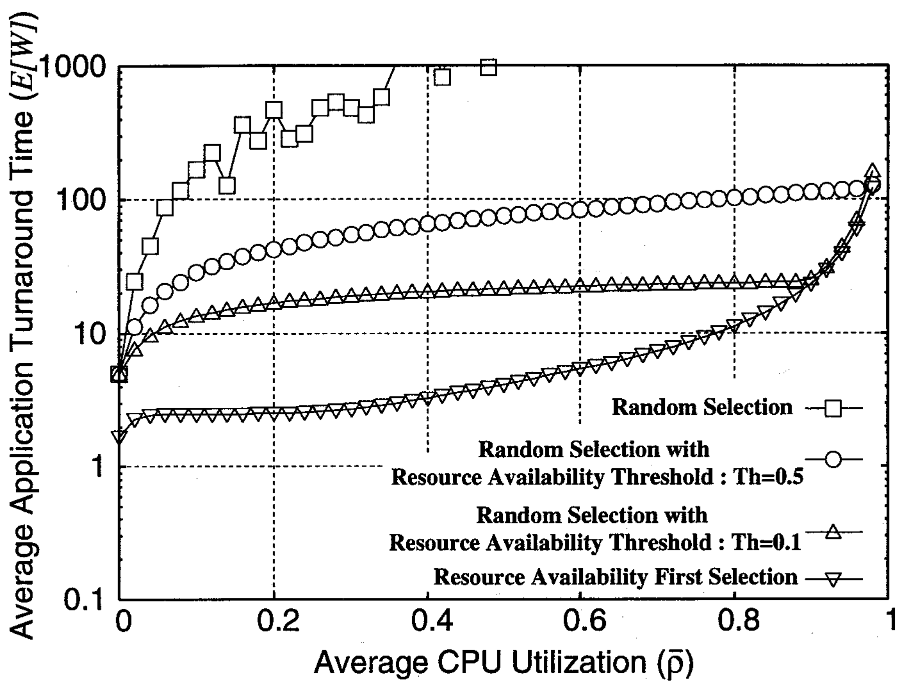

Figure 4.4: Average Application Turnaround Time vs. Average CPU Utilization.

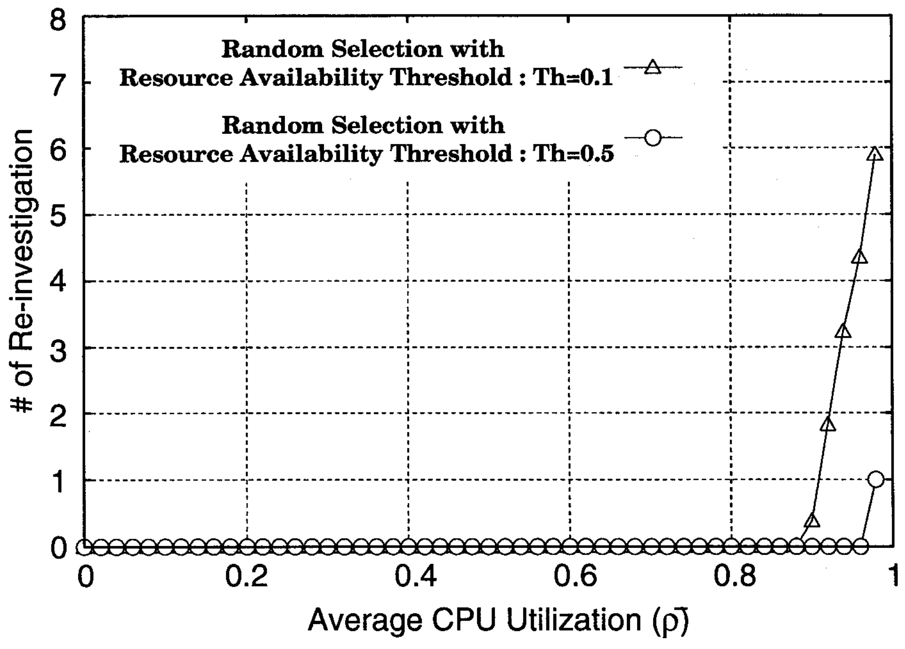

Figure 4.5: Number of Re-investigations vs. Average CPU Utilization.

puters and allocates sub-tasks accordingly. Consequently, the average application turnaround time of Scheme 2 outperforms the other schemes discussed here. In particular, when the average CPU utilization is 0, since all the sub-tasks are allocated to computers of class C, the application turnaround time becomes 1.67 (=250/150), which is much smaller than that of Scheme 1. The performance of Random Selection with Resource Availability Threshold (Scheme 3) falls between Schemes 1 and 2 in that n computers are selected randomly, but from computers that are not highly loaded. However, when the average CPU utilization is 0, the application turnaround time of Scheme 3 is the same as that of Scheme 1, because some of the sub-tasks are allocated to computers of class A. In Fig. 4.4, as the threshold Th is reduced, the average application turnaround time becomes smaller over a wide range of $\bar{\rho}$ and approaches that of Scheme 2. However, reducing Th may, in turn, result in re-investigation if too few computers are available. Although this is a concern, Fig. 4.5 shows that no re-investigation occurs under the present model until $\bar{\rho}$ becomes more than 0.9, even in the case of $Th = 0.1$.

Ratio of Sub-tasks Allocated to Computers in Each Class

Given the present method of updating computer information, it may occur that many users will select similar sets of computers based on information that may not be the latest, which could lead to critical performance degradation.

To examine this in more detail, the ratio of sub-tasks allocated to computers belonging to each class is investigated. As shown in Fig. 4.6, Scheme 2 allocates many sub-tasks to computers with higher CPU performance (i.e., class C). As the average CPU utilization increases, computers with relatively low CPU performance are also selected. However, even if the average CPU utilization is 0.5, 60% of all the sub-tasks are assigned to computers in class C. Therefore, it is very likely under Scheme 2 that many users in the Grid environment will select a few computers with high performance or lightly loaded at the same time, thereby leading to critical performance degradation if tasks are allocated independently by several

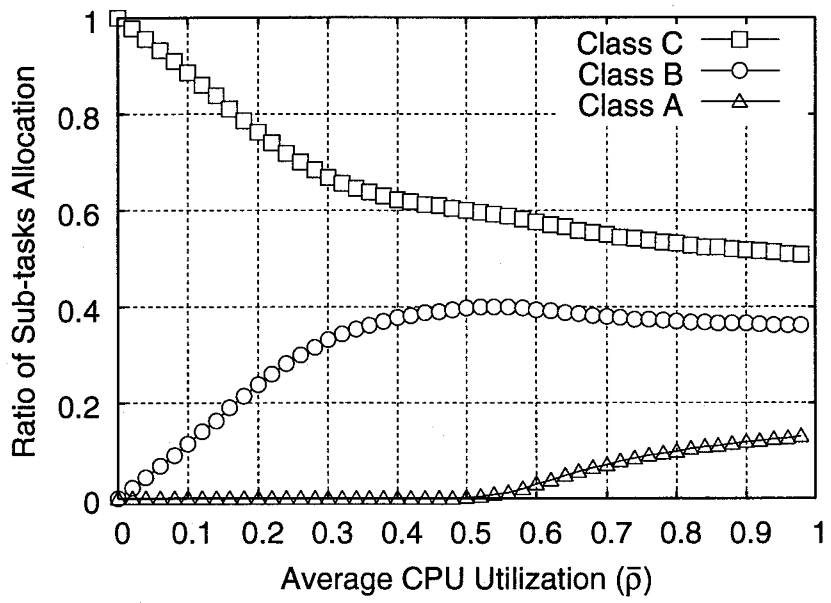

Figure 4.6: Ratio of Sub-tasks Allocated to Each Class (Scheme 2: Resource Availability First Selection).

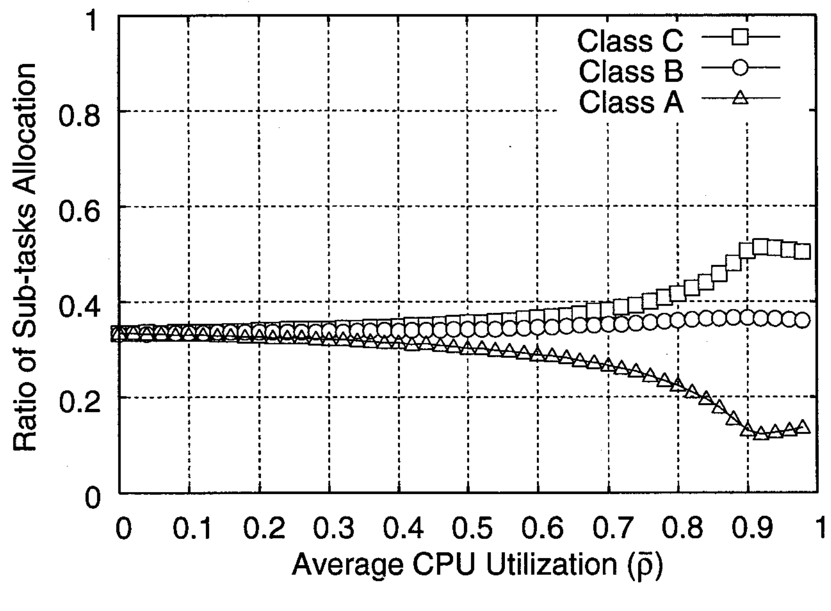

Figure 4.7: Ratio of Sub-tasks Allocated to Each Class (Scheme 3: Random Selection with Resource Availability Threshold: $Th = 0.1$).

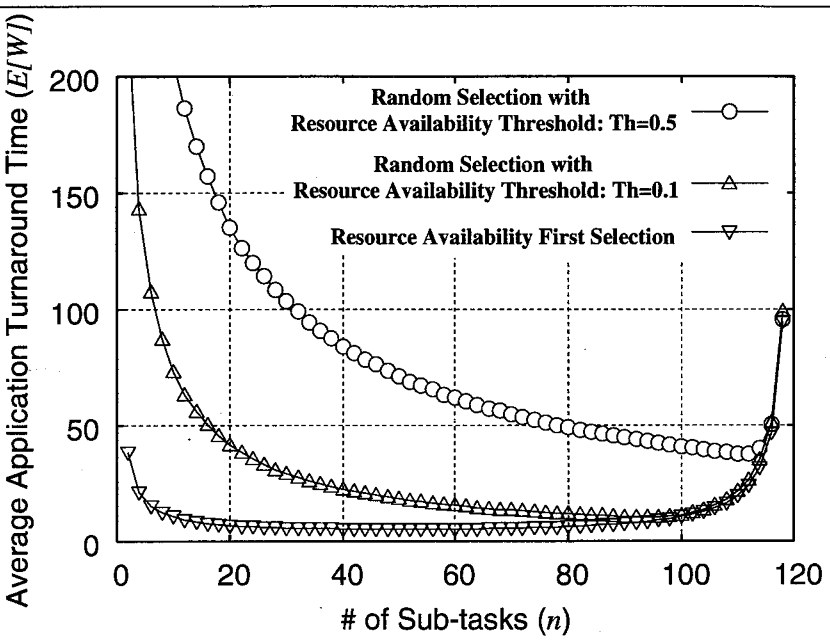

Figure 4.8: Average Application Turnaround Time vs. Number of Sub-tasks.

schedulers.

Figure 4.7 shows that Scheme 3 selects computers evenly from all classes. For example, in the case that the average CPU utilization is smaller than 0.5, the ratio of sub-tasks allocated to each class is around 0.33. Therefore, Scheme 3 will use all types of computers fairly, while providing moderate performance, as shown in Fig. 4.4.

Effect of the Number of Sub-tasks on the Average Application Turnaround Time

The effect of the number of sub-tasks is illustrated in Fig. 4.8 in terms of the average application turnaround time for average CPU utilization $\bar{\rho}$ of 0.6. Normally, as the number of sub-tasks increases, the workload of each sub-task W_s decreases, resulting in a comparable decrease in average application turnaround time for Scheme 2 down to some minimum value. As shown in Fig. 4.8, as the number of sub-tasks is increased past 50, the average application turnaround time begins to increase again, because the user must allocate sub-tasks to highly loaded computers with higher probability. The average application turnaround time for Scheme 3 improves with the number of sub-tasks, as long as all the selected computers

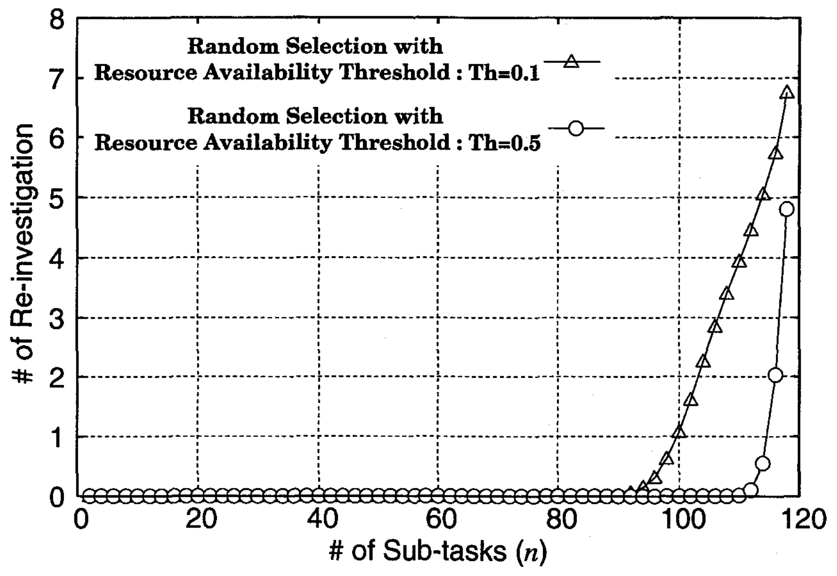

Figure 4.9: Number of Re-investigations vs. Number of Sub-tasks.

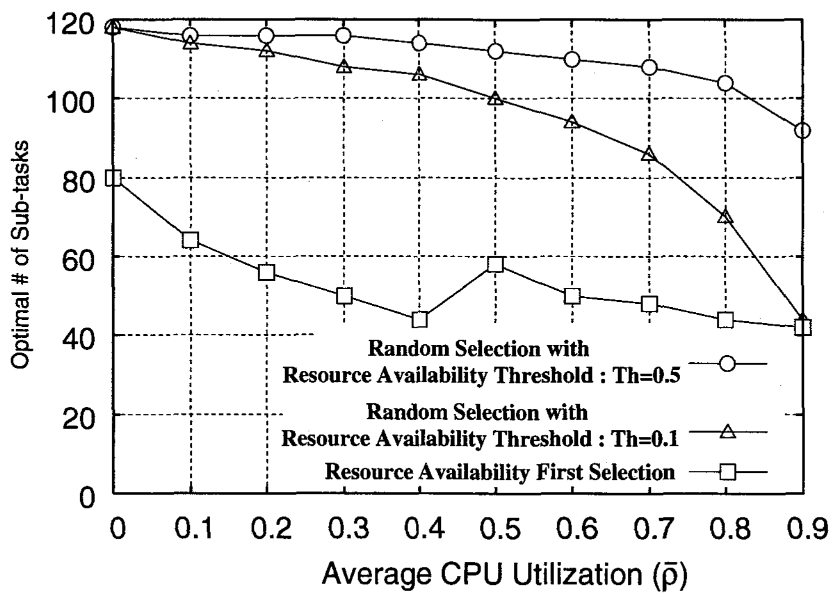

Figure 4.10: Optimal Number of Sub-tasks.

have I_i smaller than Th. In Fig. 4.8, the average application turnaround time decreases with the increase in n for $Th = 0.1$, as long as n is smaller than 94. In this case, no re-investigation occurs. However, as shown in Fig. 4.9, which illustrates the number of re-investigations in Scheme 3 as a function of the number of sub-tasks n, if n exceeds 94 and more, the number of re-investigations becomes larger. In this time $\bar{\rho}$ is fixed to 0.6 whereas $\bar{\rho}$ is smaller than 0.9, no-re-investigations occur as stated in Fig. 4.5. Therefore, re-investigations in Scheme 3 occurs depend on $\bar{\rho}$, n, and Th, and leads to the increase of the average application turnaround time.

As mentioned above, there exists an optimal number of sub-tasks that minimizes the average application turnaround time for some value of $\bar{\rho}$. Figure 4.10 illustrates the optimal number of sub-tasks for each $\bar{\rho}$. The optimal number of sub-tasks is dependent on $\bar{\rho}$, demonstrating that a larger number of sub-tasks does not always lead to an improvement in the application turnaround time. Therefore, the number of sub-tasks should be determined carefully in consideration of $\bar{\rho}$, that is, the average load imposed on the Grid computing system. This figure also shows that the optimal number of sub-tasks is larger for a higher threshold Th.

4.4 Simulation

In the previous section, I analytically evaluated the average application turnaround time by modeling each computer as an M/G/1-PS queue, and investigated the fundamental performance and characteristics of each task allocation scheme. Numerical results show that Scheme 2 outperforms other schemes presented in this chapter in terms of the application turnaround time, and selects a few computers with high performance, on the other hand, Scheme 3 can use all types of computers fairly. Thus, it may lead Scheme 2 to the critical performance degradation in the realistic Grid environment where the obtainable information on the utilization can involve some uncertainty and many schedulers independently allocate their tasks. In the rest of this chapter, I assume a realistic Grid environment and evaluate performance by computer simulation.

In order to verify the efficiency of the simple analysis, I first compare the performance obtained through the analysis with that of the simulation, in which the model is almost the same as the analytical one, except that the information related to the resource status in each computer is updated periodically. Next, assuming that all computers comprising the Grid environment also act as "users", and that they divide their own task into some sub-tasks and assign them to each other with the same task allocation scheme, I can also investigate the performance of some of the task allocation schemes adopted in the analysis.

4.4.1 Simulation Model

I evaluate the performance of the task allocation schemes in the simulation, which uses a model that is almost the same as the analytical model shown in Fig. 4.3. Some of the details in this simulation are different from the analysis. In the analysis, since the amount of task load injected by one user is small enough to be neglected compared with the utilization of the computers as stated in Section 4.3.1, I assume that the resource utilization of each computer never changes at each evaluation. However, in a realistic environment, even if there is only one user, the utilization can fluctuate by its own task. Thus, in the simulation, I assume that Grid users can receive periodically updated information related to the resource status of the computers, and the update interval in the computers becomes the important parameter to determine a magnitude of the uncertainty of the resource status. Here, I define the updated information as the average value of resource capability, as defined in Eq. (4.5), and obtained by the number of tasks in the computers during the previous update interval.

The total CPU utilization ρ_i of computer R_i can be defined by the sum of the CPU utilization ρ_{ui} associated with Grid users and ρ_i' due to tasks generated by itself. I assume that the task is generated by users according to a Poisson process at rate λ [Kle79]. Whenever one user task is generated, n sub-tasks are allocated to n computers, so that the sub-task arrival rate λ_{si} at each computer is given by Eq. (4.7). In this equation, N indicates the number of all computers in the simulation model.

$$\lambda_{si} = \frac{n\lambda}{N}. \tag{4.7}$$

Thus, the CPU utilization ρ_{ui} associated with users can be defined as follows.

$$\rho_{ui} = \frac{\lambda_{si} \times W_s}{S_i} = \frac{\lambda W}{S_i N}. \tag{4.8}$$

4.4.2 Comparison between Analytical and Simulation Results

First, I evaluate the validity of the analytical results compared with the results obtained by the simulation, in which the model is almost the same as the analytical one, except that the update interval of the resource status in each computer is set to 10 unit periods, which is a same unit as that of the application turnaround time. As in the analysis, I assume that the CPU utilization ρ_{ui} associated with tasks injected by the Grid user is small enough to be neglected compared with ρ_i' due to tasks generated by computer R_i itself, that is, I will focus on the performance of task allocation schemes in Grid environments where the load due to tasks generated by a computer itself is dominant over the entire load of the computers, as stated in Section 4.3.1. Therefore, the total CPU utilization ρ_i equals ρ_i'. Furthermore, ρ_i' is

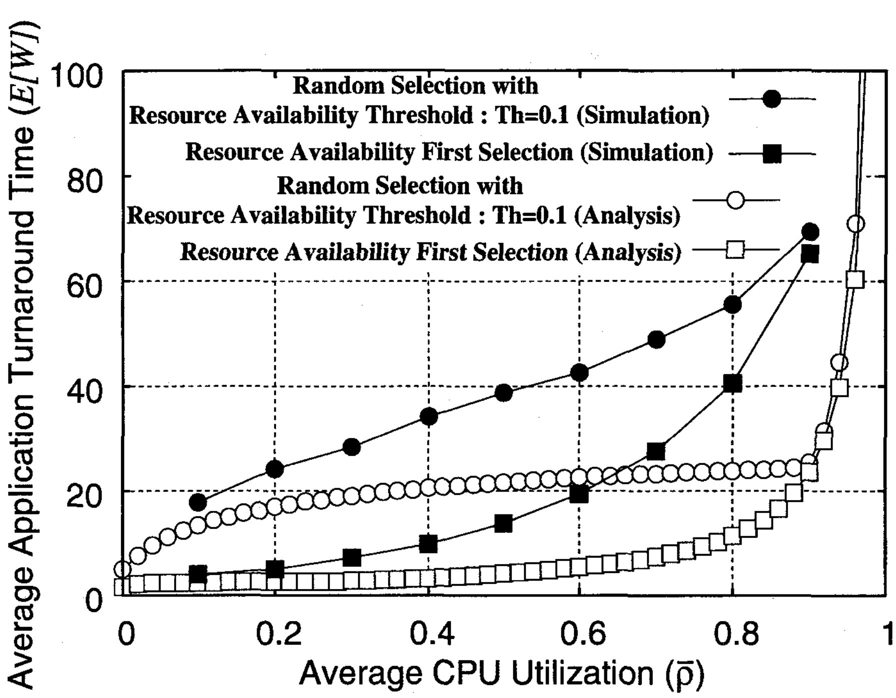

Figure 4.11: Average Application Turnaround Time vs. Average CPU Utilization.

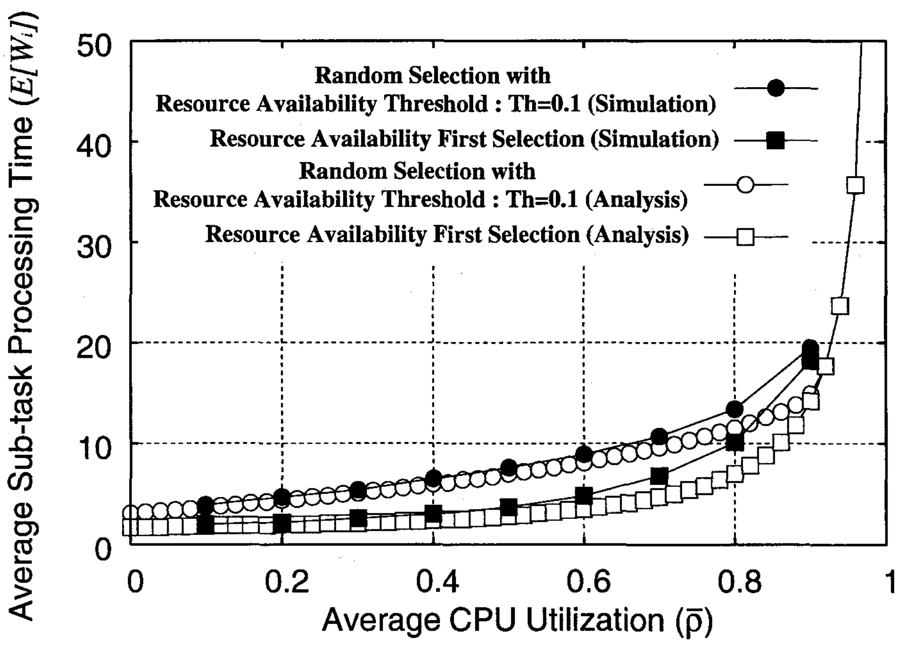

Figure 4.12: Average Sub-task Processing Time vs. Average CPU Utilization.

assumed to follow a power distribution, which is given by Eq. (4.6), based on the average value $\bar{\rho}$ for all of the computers. Thus, I obtain the performance of task allocation schemes as a function of $\bar{\rho}$. Here, the number of sub-tasks n is set to 40.

Figure 4.11 illustrates the average application turnaround time in Scheme 2 and Scheme 3 with a threshold Th of 0.1. From this figure, it is found that the average application turnaround obtained by the simulation is larger than the analytical one. This is due to some approximations employed in the analysis. First, the average application turnaround time $E[T]$ is approximately obtained by Eq. (4.3), instead of Eq. (4.2), in the analysis; the approximation in fact underestimates $E[T]$ as shown in Eq. (4.4). As a result, the analytical result does not agree with the simulation one; the former is smaller than the latter as shown in Fig. 4.11. Furthermore, the resource status of each computer is assumed to be fixed for simplicity in the analysis, whereas that actually dynamically fluctuates in the simulation. This analytical approximation also causes the difference between the analytical result and the simulation one. This can make the former larger in some cases, and smaller in other cases. The application turnaround time is, however, determined by the maximum of the sub-task processing time, as given by Eq. (4.2), so that this approximation can also make the simulation results larger than the analytical ones.

Furthermore in order to evaluate the validity of the analytical results from a different aspect, I focus on the sub-task processing time itself and the ratio of sub-task allocation to each class. Figure 4.12 shows the *average* sub-task processing time as a function of the average CPU utilization. We can see from this figure that the *average* sub-task processing time of the analysis agrees very well with the simulation over a wide range of $\bar{\rho}$. Next, from Figs. 4.13 and 4.14, which show the ratio of sub-task allocation to each class in Scheme 2 and 3, respectively, I can say that the analytical results are almost the same as the simulation results.

Hence, I can conclude that the simple analysis is very useful for evaluating many characteristics of a Grid environment in which the number of users and/or the amount of tasks

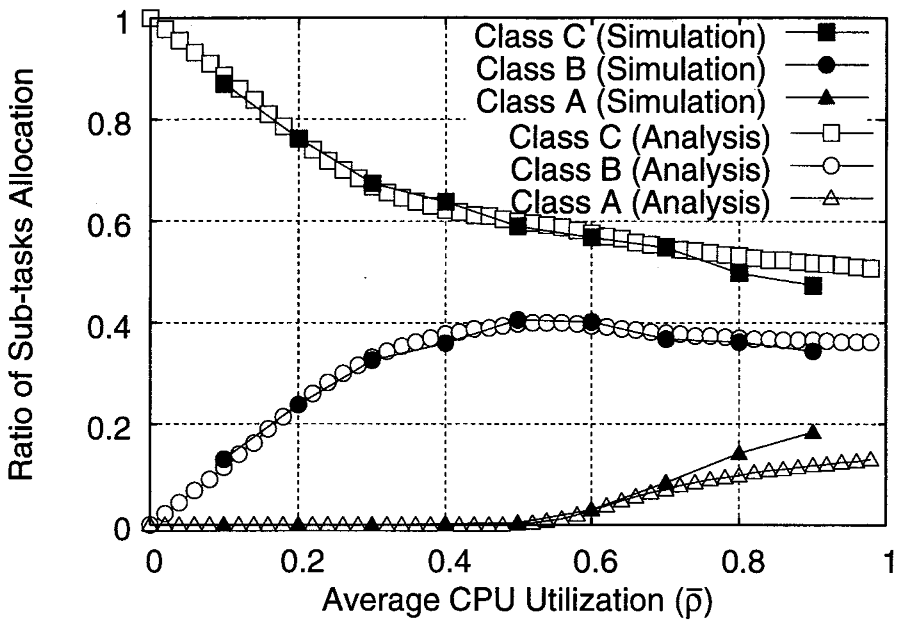

Figure 4.13: Ratio of Sub-tasks Allocated to Each Class (Scheme 2: Resource Availability First Selection).

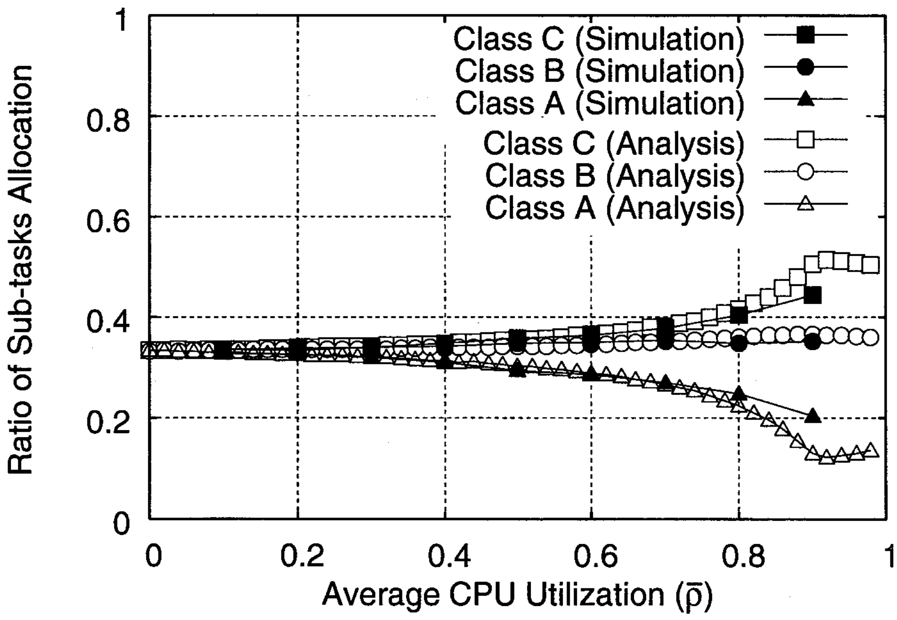

Figure 4.14: Ratio of Sub-tasks Allocated to Each Class (Scheme 3: Random Selection with Resource Availability Threshold: $Th = 0.1$).

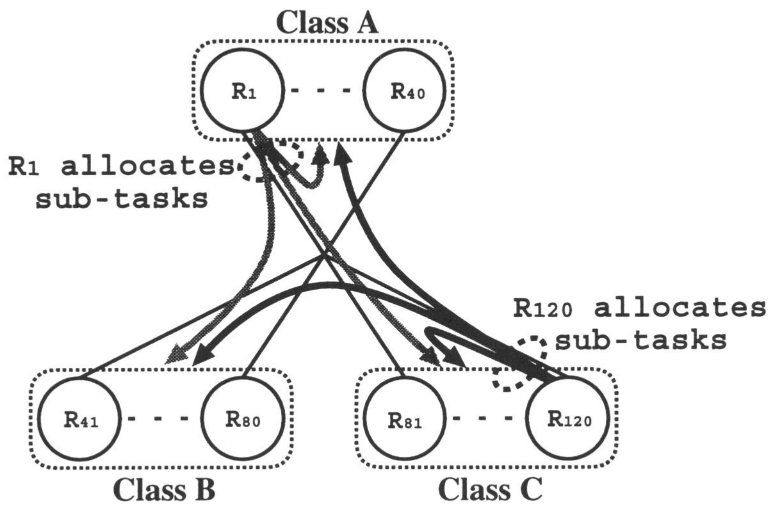

Figure 4.15: Simulation Model.

allocated to computers is relatively small. Therefore the analysis may not be applicable to a realistic environment with many users. Thus, in the next subsection, I investigate performance in a realistic environment only by simulation.

4.4.3 Performance Evaluation in the Multi-User Environment

In the realistic Grid environment, since there are many users allocating their tasks to computers and information related with resource status in each computer is updated periodically, it is likely that divided sub-tasks would concentrate on a small number of computers with relatively high utilization. This tendency may lead to a critical performance degradation of task allocation schemes. Therefore, in this subsection, I assume that computers also act as Grid users and allocate some of their tasks to each other, and I investigate the impact of the update interval on the performances of Scheme 2 and Scheme 3. The simulation environment is shown in Fig. 4.15, in which the number of computers N is set to 120, and they are evenly categorized into three classes in terms of CPU performance, namely class A, B, and C, and their CPU performance is denoted by S_A, S_B and S_C, respectively. As an example of task allocation, in Fig. 4.15, computers R_1 and R_{120} assign their tasks at almost the same time without considering the task allocation of each other, which may lead to the critical performance degradation due to the concentration of a large number of tasks on a small number of computers. Since the CPU utilization ρ_{ui}, which is associated with tasks allocated from other computers, as given by Eq. (4.8), differs among the three classes, I thus redefine the "average" CPU utilization $\bar{\rho}_u$ as follows:

$$\bar{\rho}_u = \frac{\lambda W}{N} \times \frac{1}{3}\left(\frac{1}{S_A} + \frac{1}{S_B} + \frac{1}{S_C}\right). \tag{4.9}$$

In the simulation, the CPU utilization of each computer due to tasks generated and executed by themselves, ρ_i', is set to 0.1, and $\bar{\rho}_u$ is set to 0.4, so that the total CPU utilization ρ_i of each computer is 0.5.

Figure 4.16 depicts the average and 99th-percentile application turnaround time in Scheme 2 and Scheme 3 with the threshold of 0.1, where 99th-percentile application turnaround time means that 99% of application turnaround times over all experiments distribute within this value. As shown in this figure, the application turnaround time in Scheme 3 is almost the same as that of Scheme 2, and it increases very little as the update interval in each scheme increases, whereas the 99th-percentile value of the application turnaround time in Scheme 2 increases markedly compared with that in Scheme 3. In other words, in the worst case, Scheme 2 leads to critical performance degradation as expected previously and cannot achieve the stable performance.

Figure 4.17 shows the ratio of sub-tasks allocated to computers belonging to each class as a function of the update interval of the resource information. It is found from this figure that, in Scheme 2, sub-task allocation from other computers is concentrated on a few computers belonging to class C with high CPU performance. Even if the CPU utilization of this class of computers is low at the start of an update period, these computers are likely to be allocated more sub-tasks than others with low or medium CPU performance. Remarkably, this happens when the update interval becomes large, namely the magnitude of the uncertainty of the resource status is large, and this leads to performance degradation in Scheme 2, as shown in Fig. 4.16. On the other hand, Scheme 3 allocates sub-tasks to all computers in almost

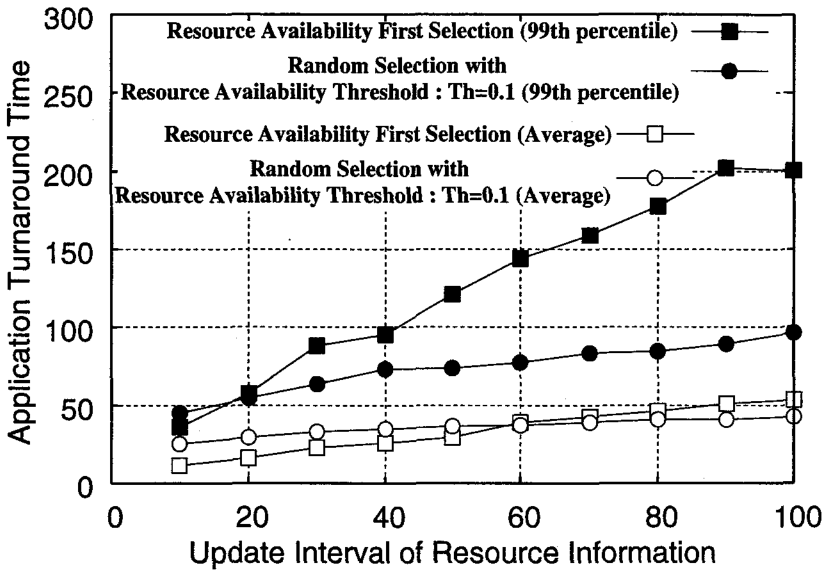

Figure 4.16: Update Interval of Resource Information vs. Average Application Turnaround Time.

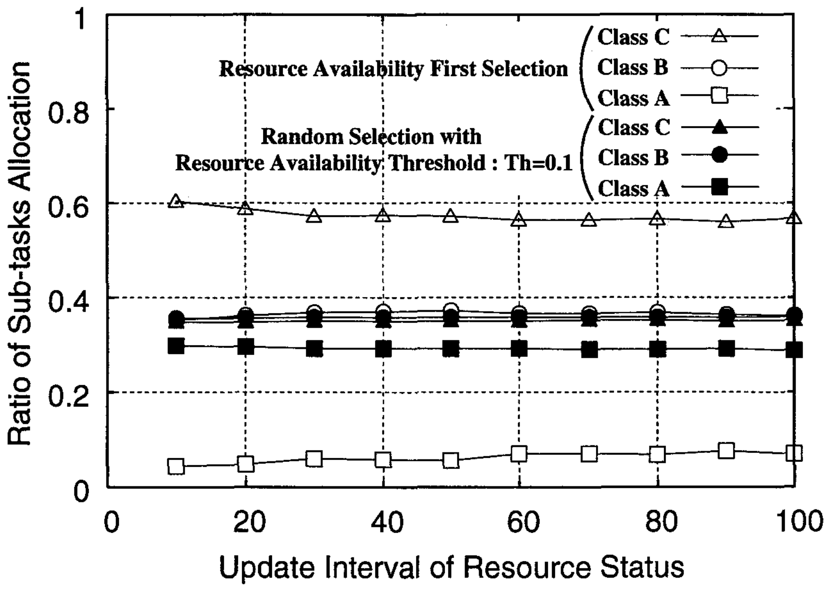

Figure 4.17: Update Interval of Resource Information vs. Ratio of Sub-tasks Allocation.

the same manner as in the analysis, so that, the resource capability does not change greatly during one update interval. Hence, Scheme 3 can achieve a moderate performance from the practical viewpoint, while Scheme 2 may degrade the performance, in particular, of the 99th-percentile application turnaround time in a realistic environment, in which the resource utilization information used for task allocation must be uncertain due to the update delay.

4.5 Conclusion

In order to examine the behavior of sub-task allocation schemes in the Grid computing environment, first I analytically evaluated performances of some task allocation schemes by modeling each computer as an M/G/1-PS queue. I adopted three task allocation schemes in this chapter: random selection of computers (Scheme 1), selection of n least-loaded computers based on the current resource availability (Scheme 2), and random selection of n computers from among them with processing capability greater than a predefined threshold (Scheme 3).

The performance of each scheme was investigated preliminarily in terms of the average application turnaround time as a function of the average CPU utilization of computers for the case in which the load due to the task allocation from a single user is very small and it does not change the long-term CPU utilization of computers. Through some numerical results, I found that Scheme 1 is not useful due to randomly chosen computers with high loads. Scheme 2 outperforms both of the other schemes, even though it causes the concentration of sub-tasks on a few computers. Scheme 3 achieves moderate performance over a wide range of average CPU utilization without concentrating task allocation.

Next, assuming the realistic Grid environment where information related with the resource status in each computer is updated periodically, I compared the performance obtained by the simulation with that of the analysis when there is a single user, and I investigated the application turnaround time of Scheme 2 and Scheme 3 when all computers act as "users" and some of their own tasks are allocated to each other. The analysis agrees with the simulation very well when there is a single user. Furthermore, when all computers act as users, Scheme 3 succeeds in handling the uncertainty imposed by both the dynamic change of resource utilization and the periodic update of resource information in a way to introduce some randomness in selecting computers, whereas Scheme 2 leads to critical performance degradation due to the concentration of task allocation with a long update period. Thus, I can conclude that Scheme 3 is applicable to a realistic Grid environment.

Chapter 5

Parallel Transferable Uniform Multi-Round Algorithm for Achieving Minimum Application Turnaround Times in Heterogeneous Distributed Computing Environments

5.1 Introduction

As the performance of wide-area networks and off-the-shelf computers increases, large-scale distributed computing has become feasible, realizing a virtual high-performance computing environment by connecting geographically distributed computers via the Internet. Such an environment has high heterogeneity in terms of computation and communication resource capacities of individual computers (i.e., the computation power of each computer and the communication bandwidth of the link attached to the computer). This type of Grid computing has recently increased in popularity for parallel applications [FK98a, FKT01]. The master/worker model is suitable for loosely coupled applications and particularly suited to Grid computing environments involving a large number of computers that have different resource capacities each other. In this model, a master with application tasks (and data to be processed) dispatches subtasks to several workers, which process the data allocated by the master. Such data transmissions are usually performed by using TCP (Transmission Control Protocol) that provides a reliable data delivery between the master and the worker. A typical instance of applications based on this model is a divisible workload application [BGMR96, Rob03], where the master divides the data to be processed into an arbitrary number of chunks and then dispatches the chunks to multiple workers processing the chunks. For a given application, it is assumed that all chunks require identical processing. This divisibility is encountered in various applications such as image feature extraction, where a complete task typically consists of many subtasks that may involve operations on pixels or clusters of pixels at a common level [GO93].

Recent applications have involved large amount of data, and the transmission time of data sent from the master to workers is no longer negligible compared to the computation time [CFK$^+$00]. Therefore, the master is required to schedule the processing workload of workers effectively by considering the impact of communication time and computation by each worker on the application turnaround time. A number of scheduling algorithms have been proposed in which the master dispatches workload to workers in a 'multiple-round' manner to overlap the time required for communication with that for computation [BGMR96, YC03b, YC02, BLR03]. These methods, such an the uniform multi-round algorithm (UMR), can mitigate the adverse effects of time spent to transmit large amounts of data, and have made it possible to achieve good application turnaround times. However, these methods assume a homogeneous network model, in which both the networks associated with the master and workers have the same transmission capacity, adopting a sequential transmission model. In this model, the master uses its network resource in a sequential manner, that is, the master transmits data to one worker at a time [CBLR02, Ros01]. In actual networks, where the master and workers are connected via a heterogeneous network, the sequential transmission model is unable to minimize the adverse effects of data transmission time on the application turnaround time. Therefore, in this study, a new scheduling algorithm called parallel transferable uniform multi-round (PTUMR) is proposed based on the UMR algorithm [YC03b, YC02]. In the PTUMR algorithm, the master divides workers into appropriate groups and transmits chunks to all workers in each group in parallel in order to treat the set of workers in a group as one virtual worker to whom the master transmits chunks in a sequential manner like in UMR. The PTUMR analytically determines how application data should be divided and when data should be sent to workers in heterogeneous environments to nearly minimize application turnaround time. Performance evaluations indicate that the algorithm can reduce the adverse effects of data transmission time on application turnaround time more thoroughly compared with the conventional UMR under various environments with a wide range of heterogeneity, allowing turnaround times close to the lower limits to be achieved.

This chapter is organized as follows. In Section 5.2, the conceptual basis for multiple-round scheduling and the conventional UMR algorithm are introduced. The proposed PTUMR algorithm is presented in Section 5.3, and its performance is investigated in Section 5.4. The chapter is finally concluded in Section 5.5.

5.2 Conventional UMR Scheduling Algorithm

The concept behind the multiple-round scheduling method and the heterogeneous distributed computing model employed in this study are introduced briefly followed along with a description of the standard UMR algorithm as an example of existing multiple-round scheduling algorithms.

5.2.1 Multiple-Round Scheduling Method

Recently, a number of scheduling methods have been proposed in which the master dispatches data to workers in a multiple-round manner in order to overlap communication with computation and thereby reduce the application turnaround time. Figure 5.1 shows an exam-

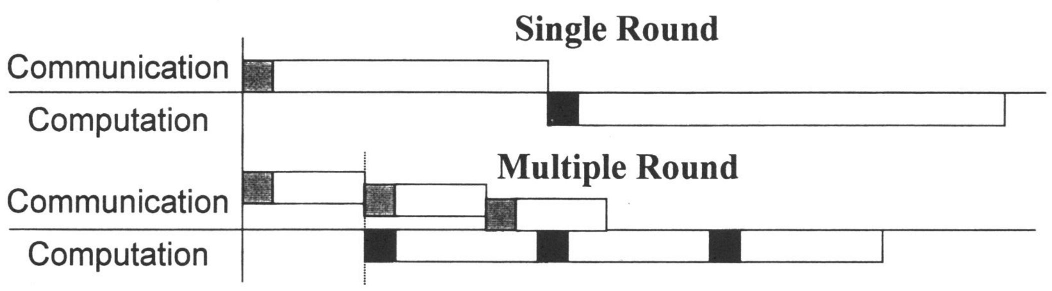

Figure 5.1: Multiple-round scheduling.

ple of this scenario, where the master dispatches a workload of the application to one worker. In single-round scheduling, the master transmits the entire set of application data W [units] to the worker at one time. In multiple-round scheduling, on the other hand, the data W is divided into multiple chunks of arbitrary size and processed in M rounds. Here, in Round $j(= 0, \cdots, M-1)$, the master allocates one chunk of C_j [units] in length to the worker. In data transmission, the master transmits data to the worker at a rate of B [units/s], and the worker computes the data allocated by the master at a rate of S [units/s]. In addition, it is assumed that the time required for the workers to return computation results to the master is small enough to be neglected. Prior to the start of data transmission, a fixed time interval independent of the size of the chunk is added due to some initial interaction between the master and workers such as TCP connection establishment. It is assumed that the size of data exchanged in this interaction is small enough so as that the overhead is also independent of the bandwidth of the network between the master and the worker. This fixed interval added at the start of data transmission is defined as ϵ [s]. Similarly, a part of the computation time is independent of the chunk size is defined as δ [s]. For example, this time includes preparation and invocation time to start the computational process on each worker. Using the above definitions, the time T_{comm_j} needed for the master to send C_j units of a chunk to a worker is formulated as follows.

$$T_{comm_j} = \epsilon + \frac{C_j}{B}. \tag{5.1}$$

The time T_{comp_j} required for the worker to compute C_j units of a chunk received in Round j is modeled as follows.

$$T_{comp_j} = \delta + \frac{C_j}{S}. \tag{5.2}$$

Multiple-round scheduling is briefly compared below with single-round scheduling in terms of the application turnaround time T_{ideal} under the ideal assumption that each worker never enters the idle computation state once the first chunk has been received. In single-round scheduling, the worker starts computation after receiving the entire set of data W from the master, as shown in Fig. 5.1. Therefore, the application turnaround time in single-round scheduling is represented as follows.

$$T_{ideal} = \epsilon + \frac{W}{B} + \delta + \frac{W}{S}. \tag{5.3}$$

In multiple-round scheduling (over M rounds), the worker can start computation after receiving only C_0 units of the data in the first round (Round 0). The application turnaround time is therefore given by

$$\begin{aligned} T_{ideal} &= \epsilon+\frac{C_0}{B}+\sum_{j=0}^{M-1}\left(\delta+\frac{C_j}{S}\right), \\ &= \epsilon+\frac{C_0}{B}+M\times\delta+\frac{W}{S}. \end{aligned} \tag{5.4}$$

In single-round scheduling, the data transmission time directly affects the application turnaround time, whereas under multiple-round scheduling, the adverse effects of data transmission time on the application turnaround time can be reduced because the data transmission time in Round 1 and later can be overlapped with the computation time. However, the use of a large number of rounds would lead to an increase in the total overhead, and eventually result in a degradation of application turnaround time. Thus, optimizing both the number of rounds and the size of chunks so as to minimize the application turnaround time is a key issue in multiple-round scheduling.

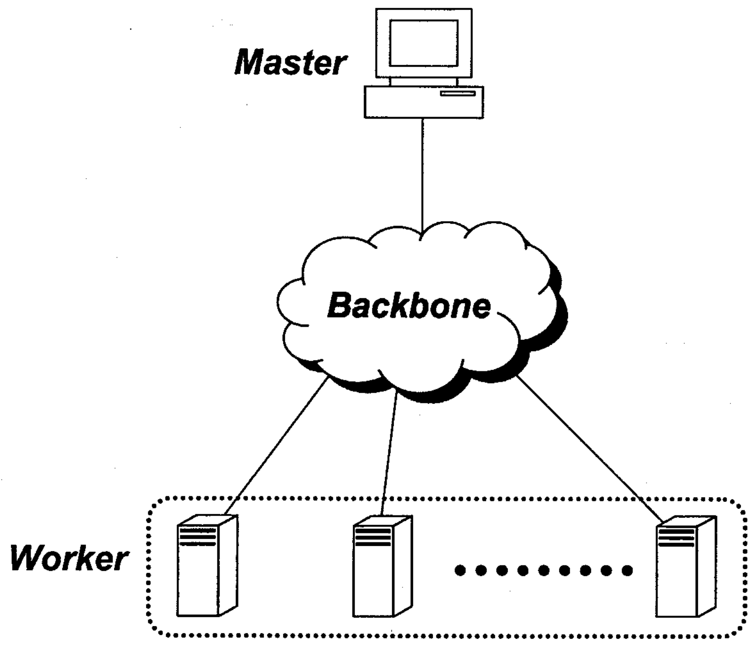

Figure 5.2: Distributed computing model.

5.2.2 Heterogeneous Distributed Computing Environment

The heterogeneous distributed computing model employed here is shown in Fig. 5.2. The master and N workers are connected to a high-speed network that is free of bottlenecks, that is, the internal links of the backbone network are assumed to have much higher transmission capacities than the links attached directly to the master and workers such that congestion on internal link is negligible.

This model has heterogeneity in terms of the computation and communication capacities of workers, namely, the computation rate S_i [units/s], the data transmission rate B_i [units/s] of the link attached to the worker, the overhead δ_i [s] added at the computation time, and the overhead ϵ_i [s] added at the data transmission time. In my study, these parameters of each worker are randomly and independently chosen according to some uniform distributions (See Section 5.4).

Furthermore, the data transmission rate of the link attached to the master is defined as

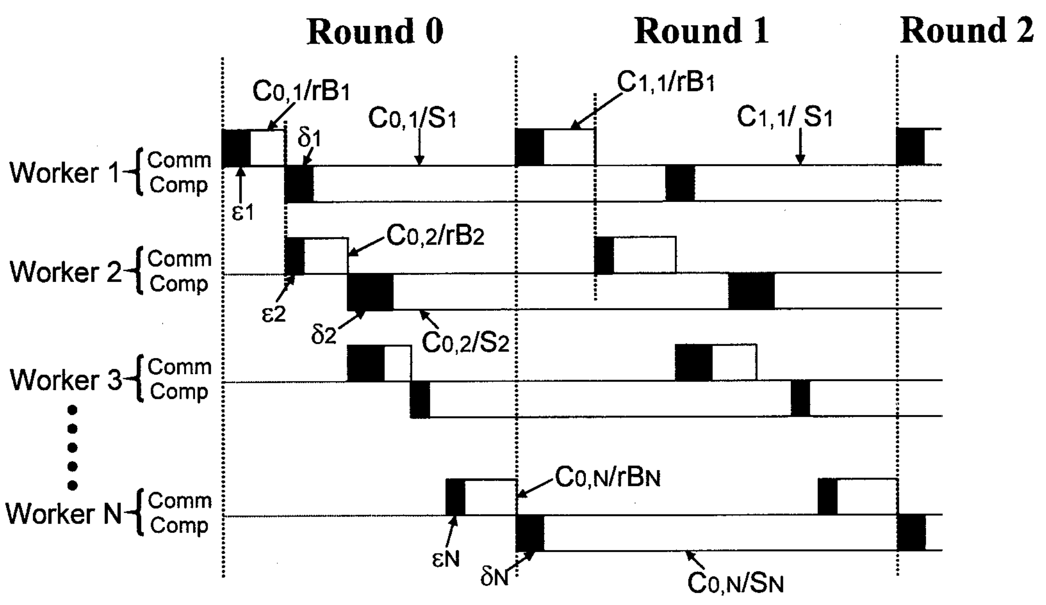

Figure 5.3: Timing chart of data transmission and worker processing under UMR.

B_0, which can be shared by multiple data transmissions between the master and multiple workers. I define the actual data transmission rate between the master and the worker i as rB_i, which may be smaller than B_i due to the sharing of master-side link capacity by multiple data transmissions.

5.2.3 UMR

UMR is an example of multiple-round scheduling algorithm [YC03b, YC02]. The distributed computing model for UMR is shown in Fig. 5.2. UMR adopts the sequential transmission model, by which the master transmits a chunk to one worker at a time. Therefore, the actual data transmission rate rB_i between the master and worker i becomes $\min\{B_i, B_0\}$. Figure 5.3 illustrates how the data is transmitted to workers and then processed under UMR, where the size of chunk allocated to the worker i in Round j is denoted by $C_{j,i}$. In heterogeneous environments in terms of resource capacities of workers, the master determines the amount of chunks allocated to each worker in a way that the computation time becomes

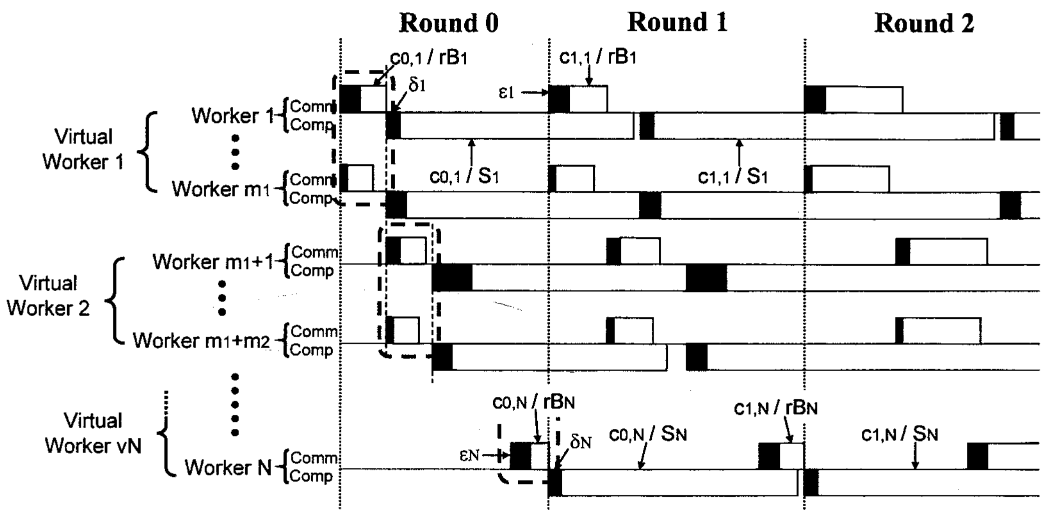

Figure 5.4: Timing chart of data transmission and worker processing under PTUMR.

identical for all workers during a round for ease of analytical approach. To reduce the data transmission time in the first round, small chunks are transmitted to workers in the first round, and the size of chunks then grows exponentially in subsequent rounds.

In UMR, the number of rounds and the chunk size in the first round have been approximately optimized under the sequential transmission model so as to mitigate the adverse effects of data transmission time on the application turnaround time [YC02]. However, the sequential transmission model adopted in UMR prevents the minimization of application turnaround time in real network environments, particularly when the transmission capacity of the master-side link is larger than that of the worker-side links. Under such conditions, the UMR algorithm cannot utilize the network resource to their maximum. Therefore, UMR requires further improvements in order to make a better use of the network resources.

5.3 PTUMR Scheduling Algorithm

In this section, the new PTUMR scheduling algorithm is presented. The PTUMR algorithm determines how the application data should be divided and when the data should be transmitted to workers in network environments that allow the master to transmit data to multiple workers in parallel by establishing multiple TCP connections. The PTUMR is based on the same distributed computing model (Fig. 5.2) as standard UMR.

In contrast to UMR, the PTUMR algorithm allows the master to transmit chunks to multiple workers simultaneously (Fig. 5.4) assuming that ϵ_i can be overlapped among concurrent transmissions so as to reduce the adverse effects of data transmission time on the application turnaround time. This extension is applicable and suitable especially for asymmetric networks, where the transmission capacity (B_0) of the master-side link is greater than that (B_i) of the worker-side links, as is often the case in actual network environments. More precisely, the PTUMR divides workers into appropriate groups to transmit chunks to all workers in each group in parallel and then treats the set of workers in each group as one virtual worker to whom the master transmits chunks in a sequential manner like in UMR, where the data transmission to one virtual worker can be equivalent to the parallel data transmissions to multiple workers composing that.

After appropriately grouping the workers, the PTUMR algorithm analytically determines the appropriate number M^+ of rounds and total size R_0^+ of the chunks allocated to all workers in the first round such that the application turnaround time for total amount W of application data is minimized under a given environment characterized by S_i, B_0, B_i, δ_i, and ϵ_i ($i = 1, 2, \ldots, N$, where N is the number of all possible workers). This algorithm is explained in further detail below.

5.3.1 Aggregation of Workers in each Parallel Transmission

The algorithm divides the workers into groups, and treats the set of workers in each group as one virtual worker. The master transmits chunks to the workers composing a virtual worker in parallel. In this subsection, I describe how to derive the resource capacity of the virtual worker based on resource capacities of workers composing the virtual worker (grouping method is presented in Subsection 5.3.3). Here, the number of workers composing virtual worker k is denoted by m_k and the number of virtual workers used for processing of the application by vN, and the set of m_k workers by L_k.

The virtual worker k computes the chunk of $C_{j,k}$, which is the total size of chunks allocated to all workers composing the virtual worker in Round j, at the rate of vS_k, with the overhead $v\delta_k$ added at the start of the computation. The computation time of virtual worker k is defined by that of the worker which finishes computing at the latest time among workers composing the virtual worker k. I assume that the computation time without the overhead δ_i added at the start of computation is identical for all workers composing the virtual worker in each round. For this assumption, the relation between size $c_{j,l}$ of chunk allocated to worker $l(\in L_k)$ in Round j and size $C_{j,k}$ of chunk allocated to the virtual worker k can be derived based on computation speed S_l of worker l and that vS_k of virtual worker k such that

$$\forall l \in L_k \quad \frac{c_{j,l}}{S_l} = \frac{C_{j,k}}{vS_k}. \tag{5.5}$$

From this equation and the assumption described above, the computation time $T_{comp_{j,k}}$ of the virtual worker k for Round j is given by

$$\forall l \in L_k \quad T_{comp_{j,k}} = \frac{c_{j,l}}{S_l} + \max_{l \in L_k}\{\delta_l\} = \frac{C_{j,k}}{vS_k} + \max_{l \in L_k}\{\delta_l\}. \tag{5.6}$$

From this equation, the largest overhead δ_l among workers composing the virtual worker k can be treated as the overhead $v\delta_k$ of virtual worker k. Furthermore, since the size $C_{j,k}$ of chunk allocated to the virtual worker k is equal to the sum of chunk size $c_{j,l}$ for all workers composing that, from Eq. (5.5), the computation speed vS_k of the virtual worker k can be denoted by the sum of S_l for all workers composing that.

Next, the virtual worker k receives the chunk of $C_{j,k}$ at the rate of vB_k, and the overhead $v\epsilon_k$ is added to the data transmission. Similarly to the computation time, I assume that the data transmission time without the overhead ϵ_i added at the start of the data transmission in each round is identical for all workers composing the same virtual worker. As shown in Eq. (5.5), the size $c_{j,l}$ of chunks allocated to worker l is determined based on computation speed S_l of the worker. Therefore, the master must transmit these chunks to worker l at the rate of rB_l ($\leq B_l$) so that the data transmission time $c_{j,l}/rB_l$ without overhead becomes identical for all workers composing the same virtual worker k. Here, the relation between the data transmission rate vB_k to virtual worker k and rB_l is given as follows.

$$\forall l \in L_k \quad \frac{c_{j,l}}{rB_l} = \frac{C_{j,k}}{vB_k}. \tag{5.7}$$

From this equation and the assumption for the data transmission, I can give the communication time $T_{comm_{j,k}}$ of the virtual worker k in Round j as follows.

$$\forall l \in L_k \quad T_{comm_{j,k}} = \frac{c_{j,l}}{rB_l} + \max_{l \in L_k}\{\epsilon_l\} = \frac{C_{j,k}}{vB_k} + \max_{l \in L_k}\{\epsilon_l\}. \tag{5.8}$$

From this equation, the largest overhead among all workers can be treated as the overhead $v\epsilon_k$ of the virtual worker k. Furthermore, since $C_{j,k}$ of the virtual worker k is equal to the sum of $c_{j,l}$ over m_k workers, from Eq. (5.7) the data transmission capacity vB_k of the virtual worker k can be denoted by the sum of rB_l for all workers composing that.

In addition, the data transmission rate vB_k between the master and the virtual worker k can be derived by solving the constraint maximization problem as follows.

$$\text{maximize } vB_k = \sum_{l \in L_k} rB_l, \tag{5.9}$$

$$\text{subject to} \begin{cases} \forall l \in L_k \quad rB_l \leq B_l, \\ \forall l \in L_k \quad \frac{S_l}{rB_l} = \frac{vS_k}{vB_k}, \\ \sum_{l \in L_k} rB_l \leq B_0. \end{cases}$$

The 2nd constraint is derived based on Eqs. (5.5) and (5.7), and implies that the master should group the workers whose computation rate S_l normalized by data transmission capacity B_l is identical. Otherwise, the actual data transmission rate rB_l of workers are limited to lower rate due to the worker with the smallest S_i/B_i, which may result in the critical degradation of the data transmission rate of the virtual worker. After the derivation of vB_k, the data transmission rate rB_l at which the master transmits the data to the worker l in L_k is derived according to Eq. (5.7).

Furthermore, from Eq. (5.6), the relation between the total size R_j of chunks allocated to all virtual workers in Round j and the size $C_{j,k}$ of chunk allocated to the virtual worker k is given by

$$\begin{aligned} C_{j,k} &= \alpha_k \times R_j + \beta_k, \\ \alpha_k &= \frac{vS_k}{\sum_{k=1}^{vN} vS_k}, \\ \beta_k &= \frac{vS_k \times \sum_{k=1}^{vN}\{vS_k \times v\delta_k\}}{\sum_{k=1}^{vN} vS_k} - vS_k \times v\delta_k. \end{aligned} \tag{5.10}$$

As shown in this equation, the size of chunk allocated to the virtual worker is proportional to its computation speed, that is, the virtual worker with higher computation speed processes larger size of chunk.

5.3.2 Derivation of Parameters Achieving Nearly Minimum Turnaround Time

The algorithm determines the number M^+ of rounds and total chunk size R_0^+ at the first round that are nearly optimal in terms of minimizing the application turnaround time under the network model where the master transmits chunks to virtual workers in the sequential manner like in UMR. In general, virtual workers may enter a computationally idle state while waiting for transmission of the next chunk to complete. Thus the application turnaround time T_{real} under PTUMR can be obtained as illustrated in Appendix A for given parameters (the number M of rounds, chunk size $R_0, R_1, \cdots, R_{M-1}$). In addition, since T_{real} is difficult to express analytically, instead of T_{real}, the ideal application turnaround time T_{ideal}, which can be readily represented in analytical form, is derived under the ideal assumption that no virtual worker ever enters the idle computation state once it has received its first chunk of data (some workers composing the virtual worker enter the computation idle state). Note that this assumption is likely to be valid when T_{real} is at its minimum. Furthermore, I also assume that the time required to compute chunks received in each round is identical for all virtual

```
1.  $Int(M)$
2.  {
3.      $T_{real_{floor-1}} = T_{real}(\lfloor M \rfloor - 1,\ R_0(\lfloor M \rfloor - 1));$
4.      $T_{real_{floor}} = T_{real}(\lfloor M \rfloor,\ R_0(\lfloor M \rfloor));$
5.      $T_{real_{ceil}} = T_{real}(\lceil M \rceil,\ R_0(\lceil M \rceil));$
6.      $T_{real_{ceil+1}} = T_{real}(\lceil M \rceil + 1,\ R_0(\lceil M \rceil + 1));$
7.
8.      $T_{real} = T_{real_{floor-1}};$
9.      $M^+ = \lfloor M \rfloor - 1;$
10.     $if(T_{real_{floor}} < T_{real})\{$
11.         $T_{real} = T_{real_{floor}};$
12.         $M^+ = \lfloor M \rfloor;$
13.     }
14.     $if(T_{real_{ceil}} < T_{real})\{$
15.         $T_{real} = T_{real_{ceil}};$
16.         $M^+ = \lceil M \rceil;$
17.     }
18.     $if(T_{real_{ceil+1}} < T_{real})\{$
19.         $T_{real} = T_{real_{ceil+1}};$
20.         $M^+ = \lceil M \rceil + 1;$
21.     }
22.     $return\ M^+;$
23. }
```

Figure 5.5: Procedure for determining the integer number of rounds.

workers, which helps an analytical approach for derivation of the appropriate parameters of the algorithm.

From this assumption, the master should complete the transmission of chunks to all virtual workers for computation in Round $j + 1$, just before virtual worker vN completes the processing of the chunk received in Round j. This situation is formulated as follows.

$$\begin{aligned}\sum_{k=1}^{vN} T_{comm_{j+1,k}} &= T_{comp_j}, \\ \sum_{k=1}^{vN} \frac{C_{j+1,k}}{vB_k} + \sum_{k=1}^{vN} v\epsilon_k &= \frac{R_j}{\sum_{k=1}^{vN} vS_k} + \frac{\sum_{k=1}^{vN}\{vS_k \times v\delta_k\}}{\sum_{k=1}^{vN} vS_k}.\end{aligned} \tag{5.11}$$

The left-hand side of the equation above represents the time required for the master to transmit the chunks for Round $j + 1$ to all virtual workers, and the right-hand side represents the time required for the virtual worker vN to compute the chunk received in Round j. From Eq. (5.11), the desired total chunk size R_j for Round j can be determined by the total chunk size R_0 in the first round as follows.

$$\begin{aligned}&R_j = \theta^j(R_0 - \gamma) + \gamma, \\ &\theta = \frac{1/\sum_{k=1}^{vN} vS_k}{\sum_{k=1}^{vN}\{\alpha_k/vB_k\}}, \\ &\gamma = \frac{\sum_{k=1}^{vN}\{vS_k \times v\delta_k\}/\sum_{k=1}^{vN} vS_k - \sum_{k=1}^{vN}\{\beta_k/vB_k\} - \sum_{k=1}^{vN} v\epsilon_k}{\sum_{k=1}^{vN}\{\alpha_k/vB_k\} - 1/\sum_{k=1}^{vN} vS_k}.\end{aligned} \tag{5.12}$$

On the other hand, since the sum of chunks allocated to all virtual workers should be equal to the total workload (application data size), the relation between the total chunk size R_0 in the first round and the number M of rounds is given by

$$\sum_{j=0}^{M-1} R_j = \frac{(R_0 - \gamma)(1 - \theta^M)}{1 - \theta} + M\gamma = W. \tag{5.13}$$

Using this relation, the application turnaround time T_{ideal} under this ideal assumption can be derived, which can be regarded as a function of the number M of rounds, as follows.

$$\begin{aligned} T_{ideal} &= \frac{1}{\sum_{k=1}^{vN} vS_k} \left\{ W + M \times \sum_{k=1}^{vN} (vS_k \times v\delta_k) \right. \\ &\left. + \sum_{k=1}^{vN} \left[vS_k \times \sum_{K=1}^{k} \left(\frac{\alpha_K \times \left(\frac{1-\theta}{1-\theta^M} \times (W - M\gamma) + \gamma \right) + \beta_K}{vB_K} + v\epsilon_K \right) \right] \right\}. \end{aligned} \tag{5.14}$$

The derivation process of the application turnaround time T_{ideal} is presented in Appendix B.

Let M^* denote the optimal number of rounds in terms of achieving the minimum application turnaround time, which can be obtained by solving $\frac{\partial T_{ideal}}{\partial M} = 0$. Then, since the number of rounds used in PTUMR should be an integer, it is necessary to determine an appropriate number of rounds M^+ as an integer and the corresponding initial chunk size R_0^+ satisfying Eq. (5.13), if M^* is not an integer. Note that, supposing Eq. (5.12) holds, the application turnaround time $T_{real}(M, R_0)$ can be calculated as a function of the given total chunk size R_0 in the first round and the number M of rounds. Therefore, as shown in Fig. 5.5, the best M^+ of four integers near M^*, $\lfloor M^* \rfloor - 1$, $\lfloor M^* \rfloor$, $\lceil M^* \rceil$, and $\lceil M^* \rceil + 1$, is chosen by comparing T_{real} corresponding to each of them. Here, $R_0(M)$ denotes the total chunk size in the first round corresponding to the number M of rounds so as to satisfy the Eq. (5.13).

Let M^+ denote the number of rounds obtained using the procedure shown in Fig. 5.5, and let $R_0^+ = R_0(M^+)$ be the total chunk size in the first round. In general, since $T_{real}(M^+, R_0^+)$ is not equal to $T_{ideal}(M^+)$ and is likely to be greater than $T_{ideal}(M^*)$, (M^+, R_0^+) is not ensured to be optimal in terms of minimizing the application turnaround time T_{real}. However, as indicated in Section 5.4, $T_{real}(M^+, R_0^+)$ may be quite close to the theoretical lower bound of T_{bound} that would be obtained by assuming an infinite capacity for all communication links.

5.3.3 Grouping Method

The proposed algorithm, PTUMR, groups the workers and treats the set of workers in the same group as one virtual worker. As described in Subsection 5.3.1, since the resource capacity of the virtual worker depends on that of each worker composing the virtual worker, the grouping method strongly affects the performance of PTUMR. In this subsection, I propose the grouping method by considering the advantage of the parallel data transmission to multiple workers demonstrated in my previous study. The details of this method are explained below.

First, all workers are sorted in ascending order of $r_i = S_i / \min(B_0, B_i)$ for grouping workers whose r_i are close to each other not to degrade their data transmission rate vB_k derived according to Eq. (5.15). The workers are put into one group up to the maximum number of workers satisfying the following equation because the master-side link capacity B_0 should be fully utilized to mitigate the adverse effect of data transmission time on the application turnaround time.

$$\sum_{l=1}^{m_k} B_l \leq B_0. \tag{5.15}$$

On the other hand, the overlap the overheads ϵ_l at the start of the data transmission for multiple workers can also mitigate the adverse effect of the data transmission, even when the data transmission capacity B_0 of the master-side link is close to that B_l of the worker-side link. Therefore, I select additional x workers to the firstly selected workers (according to Eq. (5.15)) for the group k. Note that the sum of the number of firstly selected workers and x (the number of additional ones) is referred to as m_k defined in Section 5.3.1. On the other hand, increasing x reduces the transmission speed for each worker due to sharing of the transmission capacity of the master-side link by multiple data transmissions. Therefore, there may exist an optimal number x of additional workers that minimizes the application turnaround time (no covered in this chapter). This grouping is performed repeatedly for subsequent groups until all workers are selected. In this method, I assume that x is identical for all group.

The set of workers in the same group are treated as one virtual worker k to whom the computation rate vS_k, data transmission rate vB_k, and latency parameters $v\delta_k$ and $v\epsilon_k$ can be derived as described in Section 5.3.1.

After this virtualization, the virtual workers to be used by the master for processing the application should be determined (i.e., limited) so as to prevent the performance of PTUMR from degrading due to allocation of the workload to the virtual worker with too low capacity. In this chapter, I employ the resource selection scheme presented in [YC02], which sorts virtual workers according to $vr_k = vS_k/vB_k$, and selects vN' virtual workers out of vN satisfying the following equation.

$$\sum_{k=1}^{vN'} vr_k < 1. \tag{5.16}$$

By preferentially selecting virtual workers with higher transmission rate rather than faster computation speed according to the above equation, the master can transmit larger sized chunks to virtual workers during their computation. Therefore, under the resource selection, the data transmission can be overlapped with the computation effectively, which can mitigate the adverse effect of the data transmission time on the application turnaround time.

5.4 Performance Evaluation

In this study, the distributed computing environment is heterogeneous in workers' resource capacities, and I assume that the computation rate S_i, worker-side link capacity B_i, and latency parameters ϵ_i and δ_i corresponding to related overhead of workers follow the uniform distribution. The range of the distribution is decided based on the parameter *het* which represents heterogeneity of each resource capacity in the environment, where I employ a coefficient of variation of each resource capacity as *het*. Resource capacity of each worker is randomly selected over the range as follows.

$$\left((1 - \sqrt{3} \times het) \times mean, (1 + \sqrt{3} \times het) \times mean\right). \tag{5.17}$$

where *mean* can be set to the average capacity over all workers, namely $\bar{S}$, $\bar{B}$, $\bar{\delta}$, and $\bar{\epsilon}$ listed in Tab. 5.1.

The performance of the PTUMR algorithm is evaluated in terms of application turnaround time $T_{real}(M^+, R_0^+)$ for the number M^+ of rounds and the total chunk size R_0^+ for Round 0,

Table 5.1: Model parameters and their values examined in performance evaluation.

W	$100, 500, 1000, 5000, 10000$ [units]
N	$1, 2, \cdots, 100$
$\bar{S}$	1 [units/s]
B_0	$200, 400, \cdots, 2000$ [units/s]
$\bar{B}$	200 [units/s]
$\bar{\epsilon}$	$0.001, 0.005, 0.01, 0.05, 0.1$ [s]
$\bar{\delta}$	0.1 [s]

which are derived in a way explained in the previous section. The effectiveness of PTUMR is evaluated by comparing the achievable turnaround time T_{real} with the lower bound T_{bound}, which corresponds to the best possible application turnaround time in an environment where the network resources are sufficient to render the data transmission time negligible, and any latency corresponding to related overhead is not added. In this environment, the data transmission rate B_i is close to infinity and the overheads δ_i and ϵ_i are close to 0 for all workers, so that, the single-round scheduling ($M = 1$) achieves the minimum application turnaround time as shown in Eq. (5.14). Therefore, from Eq. (5.14), T_{bound} is obtained as follows.

$$T_{bound} = \frac{W}{\sum_{k=1}^{\nu N} \nu S_k} = \frac{W}{\sum_{i=1}^{N} S_i}. \tag{5.18}$$

In the heterogeneous environment, resource capacities S_i, B_i, δ_i and ϵ_i of each worker change at every trial, so that, 100 experiments are conducted under the condition that the average resource capacity $\bar{S}$, $\bar{B}$, $\bar{\delta}$, $\bar{\epsilon}$ for all workers are fixed. And then, the mean of the application turnaround time T_{real} and that normalized by the lower bound T_{bound} for all experiments are treated as a performance measure of scheduling algorithms.

The impact of computational and network resources on the application turnaround time T_{real} is investigated below, and the performance of PTUMR is compared to that of UMR using the parameters listed in Tab. 5.1. The appropriate scale of applications for the proposed algorithm is then determined based on the total amount of application data. In these evaluation, all resource capacities, S_i, B_i, δ_i, and ϵ_i, of each worker are randomly chosen according to the uniform distribution described above, and the heterogeneity *het* of my evaluation model is first set to some value indicating moderate heterogeneity, say $\frac{3}{4\sqrt{3}}$. At last, the effect of the heterogeneity *het* on the performance of scheduling algorithms is evaluated in further details.

5.4.1 Impact of computational and network resources on application turnaround time

The impact of the computational and network resources on the average application turnaround time is examined here by assuming a total size W of 1000. In this evaluation, the number x of additional workers in each group is set to 2 or 10 (Optimal number of additional workers is not covered in this chapter). Note that the additional workers are the workers further added to the group after the master-side bandwidth can be already fully utilized, as defined in Section 5.3.3. First, I investigate the impact of the number of workers on the performance of scheduling algorithms in my evaluation model assuming a master-side link capacity B_0 of 1000. Figure 5.6 shows the application turnaround time T_{real} as a function of the number N of workers. As N increases, the application turnaround times under all scheduling algorithms presented in this chapter, PTUMR with x of 2, that with x of 10, and UMR, remarkably decrease. This is because, with the increase in the number of workers, the size of chunks allocated to each worker decreases, which further reduces the computation time on each worker. On the other hand, Fig. 5.7 shows the average normalized turnaround time T_{real}/T_{bound} as a function of N in order to compare the performance of the scheduling algorithms in further details. From this figure, you can see that when N is large, PTUMR can hold the increase of the normalized turnaround time by transferring chunks to multiple workers to utilize the network resources at master-side to their maximum, while the difference between T_{real} of UMR and T_{bound} increases with N. Furthermore, when the number x of additional workers is set to 10, PTUMR achieves T_{real} close to the lower bound T_{bound} for any N by

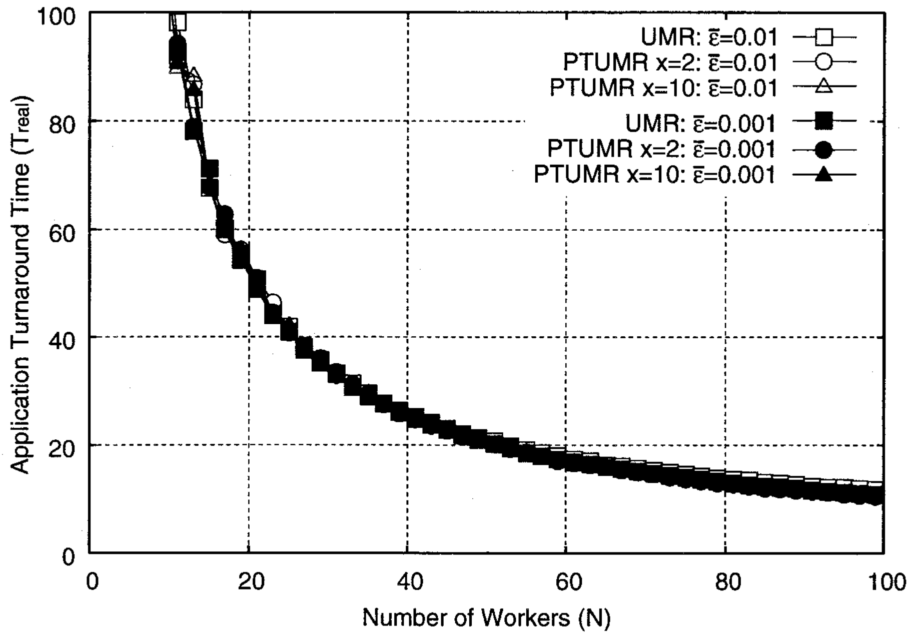

Figure 5.6: Impact of number of workers on application turnaround time.

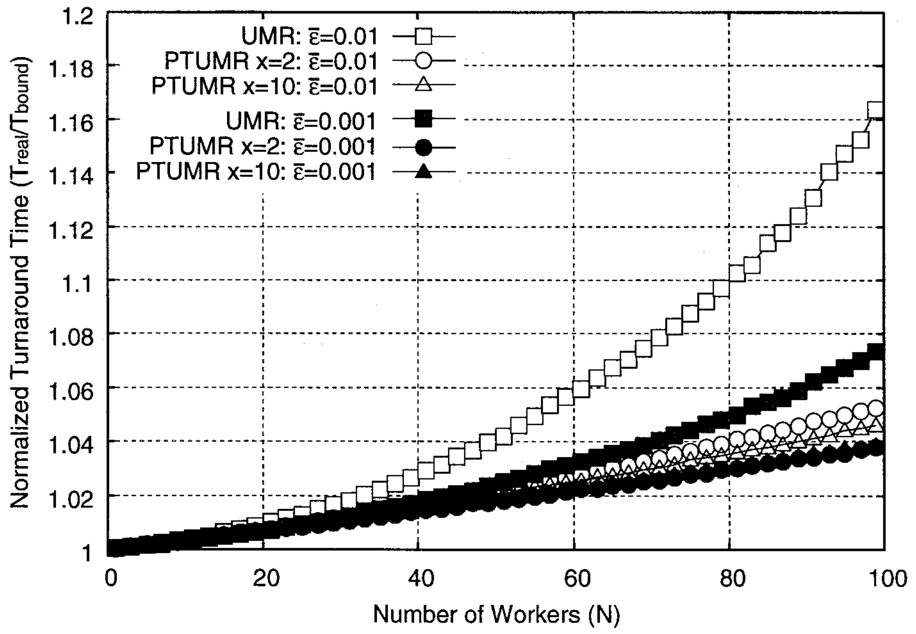

Figure 5.7: Impact of number of workers on normalized turnaround time.

aggressively overlapping the overhead δ_i added at the start of data transmission to multiple workers.

Furthermore, the performance of the PTUMR algorithm is evaluated in more detail below in reference to Fig. 5.8 and 5.9, which show the effect of B_0 and the average value of ϵ_i on performance for 100 workers (N). Figure 5.8 shows the application turnaround time T_{real} normalized by the lower bound T_{bound} as a function of the transmission capacity B_0 on the master-side link. The sequential transmission model in UMR limits the transmission speed between the master and each worker to the capacity B_i of worker-side link. Therefore, even if B_0 increases, the UMR algorithm cannot effectively utilize the network capacity, and thus cannot improve the application turnaround time. By contrast, the application turnaround time under both PTUMR with x of 2 and that with x of 10 decreases with increasing B_0 because the algorithm can utilize the full capacity by transmitting chunks to multiple workers in parallel. However, after B_0 exceeds 1000, PTUMR does not provide any further noticeable reduction in the application turnaround time, because the adverse effect of the data transmission time on the application turnaround time becomes extremely small when the data transmission capacity is much large.

Figure 5.9 shows the relationship between the average overhead $\bar{\epsilon}$ at the start of data transmission and the application turnaround time T_{real} normalized by the lower bound T_{bound}. PTUMR overcomes UMR for any $\bar{\epsilon}$ in terms of the normalized turnaround time, and especially when x is set to 10, PTUMR achieves T_{real} close to the lower bound T_{bound} for a wide range of overhead $\bar{\epsilon}$. This is because PTUMR can reduce the impact of the overhead on the application turnaround time by aggressively overlapping the overhead ϵ_i for multiple workers.

This evaluation demonstrates that the PTUMR algorithm can achieve application turnaround time quite close to the lower bound through effective utilization of the transmission capacity on the master-side and overlapping of the overhead for multiple workers.

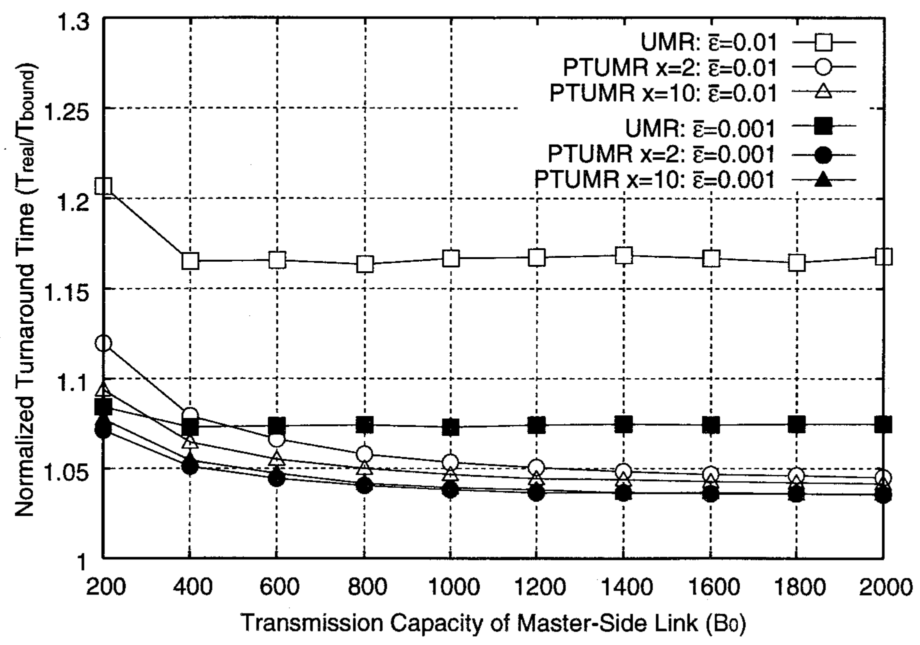

Figure 5.8: Impact of transmission capacity of master-side link on normalized turnaround time.

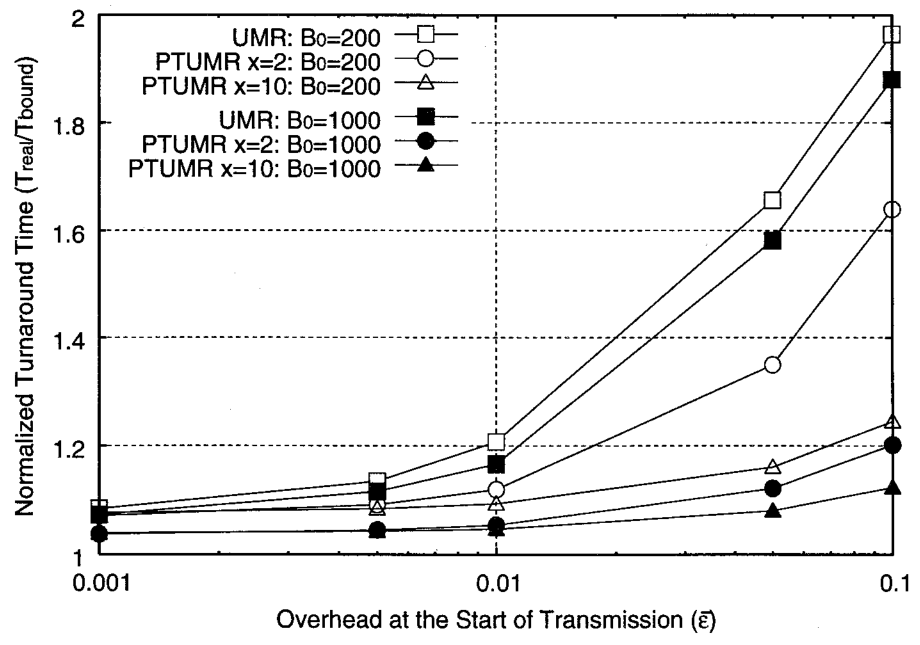

Figure 5.9: Impact of overhead at start of data transmission on normalized turnaround time.

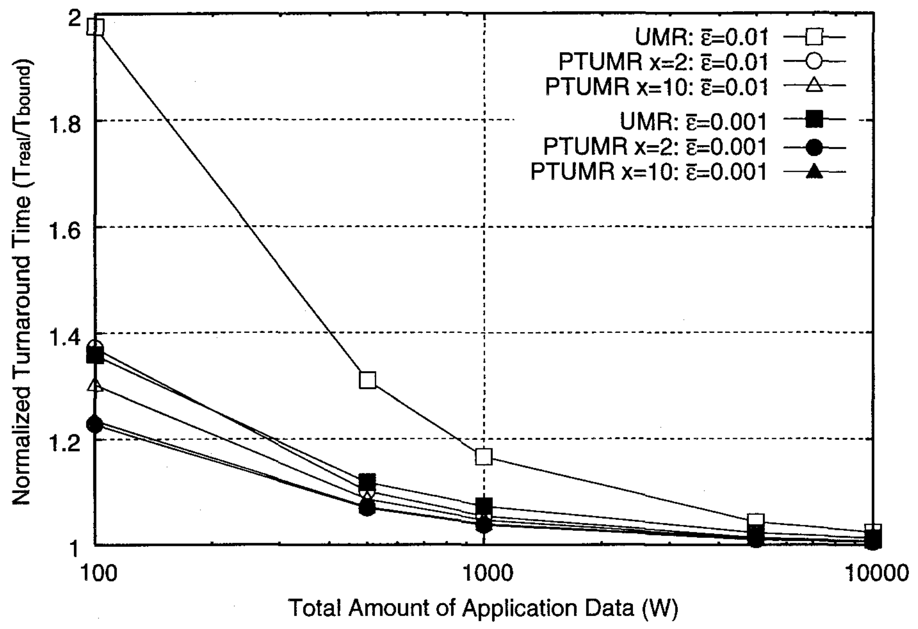

Figure 5.10: Impact of total workload (data size) on normalized turnaround time.

5.4.2 Impact of total workload on application turnaround time

Next, the effect of the total amount W of application data on the application turnaround time T_{real} is evaluated assuming a master-side transmission capacity B_0 of 1000 and 100 workers (N). Figure 5.10 shows the application turnaround time T_{real} normalized by the lower bound T_{bound} as a function of the application data size W. Under PTUMR, I set the number x of additional workers to 2 or 10. Both PTUMR with x of 2 and that with x of 10 provide excellent performance quite close to the lower bound for any W and any $\bar{\epsilon}$, that is, the PTUMR algorithm effectively eliminates the performance degradation associated with these factors. Under UMR, the normalized turnaround time becomes quite poor as the total data size W decreases, although good performance is achieved at large W. The degradation of performance at low W under UMR can be attributed to the increase in overhead ratio by decreasing the data size, which can be overcome by PTUMR. These results therefore show that the PTUMR algorithm can effectively schedule applications of any size by minimizing the adverse effects of overhead on the application turnaround time.

5.4.3 Impact of heterogeneity of the resource capacity of workers on application turnaround time

Finally, I investigate the effect of heterogeneity *het*, which is the coefficient of variation of each resource capacity of workers, on the performance of scheduling algorithms in my evaluation model assuming the total amount W of application data of 1000, a master-side transmission capacity B_0 of 1000, the average overhead $\bar{\epsilon}$ at the start of the data transmission of 0.01, and 100 workers (N). When I evaluate the impact of the heterogeneity *het* of each resource capacity, one of resource capacities S_i, B_i, δ_i and ϵ_i of each worker is randomly chosen according to the uniform distribution with the heterogeneity *het*, and others are set to average values, $\bar{S}$, $\bar{B}$, $\bar{\delta}$, and $\bar{\epsilon}$.

Figures 5.11 and 5.12 show the average normalized turnaround time T_{real}/T_{bound} and the maximum one for 100 experiments as a function of the heterogeneity *het*, where the number x of additional workers under PTUMR is set to 10. In these figures, the data points entitled 'All' correspond to the results in the condition where all recourse capacities of each worker are randomly selected according to the uniform distribution with *het*. As shown in Fig. 5.11, for a wide range of *het* under the condition where all resource capacity are randomly chosen, PTUMR overcomes conventional UMR in terms of the average normalized turnaround time. However, when the heterogeneity is large, the application turnaround time in the worst case of PTUMR becomes larger than that of UMR. With the increase in the heterogeneity of resource capacities of workers, the set of workers composing the virtual worker may include some workers with much lower resource capacities than others, which may lead to critical degradation of the resource capacity of the virtual worker as mentioned in Section 5.3.1. Therefore, the application turnaround time in the worst case of PTUMR markedly increases with *het*, and when the heterogeneity is large, PTUMR may lead to critical performance degradation compared with UMR.

On the other hand, as the heterogeneity in the worker-side link capacity B_i increases, the average performance and the worst one under both UMR and PTUMR degrades remark-

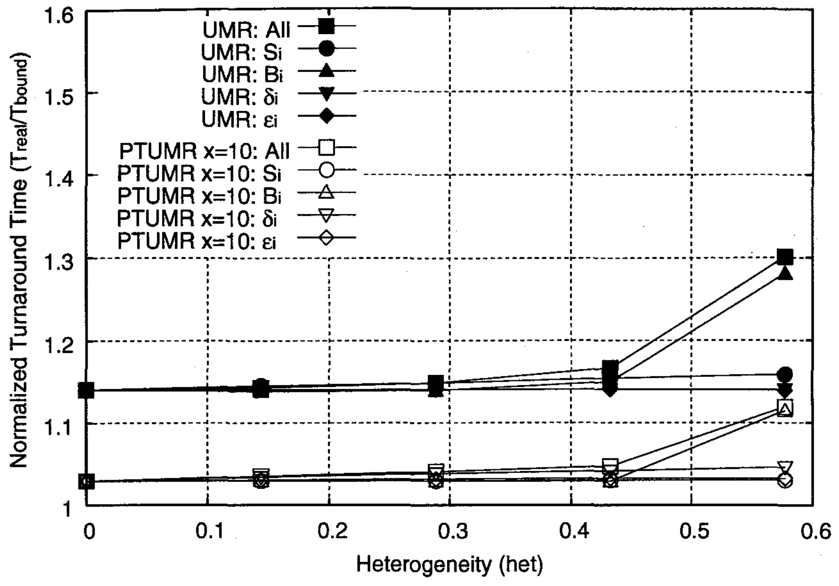

Figure 5.11: Impact of heterogeneity of each resource capacity on average normalized turnaround time

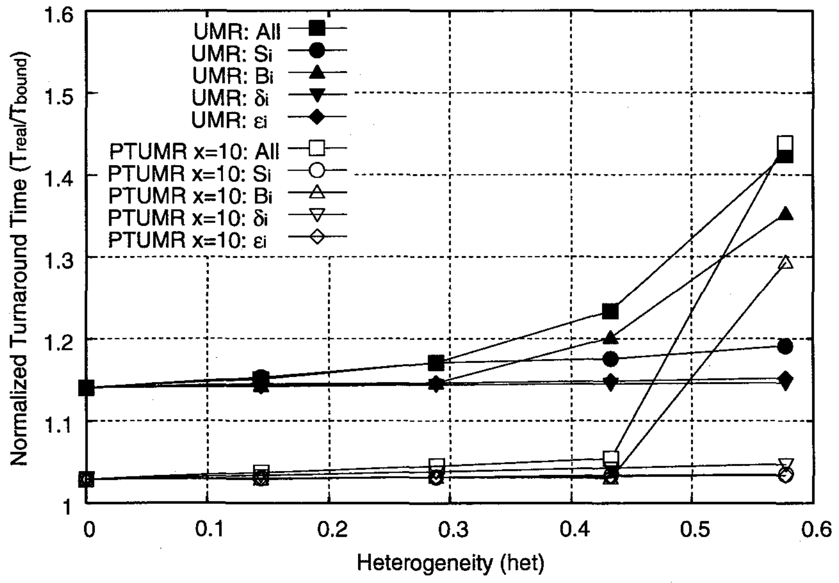

Figure 5.12: Impact of heterogeneity of each resource capacity on maximum normalized turnaround time

ably as well, while the performance is insensitive to the heterogeneity of other capacities of the worker. Therefore, I can say that the effect of heterogeneity of B_0 is dominant in the heterogeneous distributed computing model.

To hold the degradation of the resource capacities of virtual workers, I present one example of a modified PTUMR which selects workers whose resource capacities are close to each other. This modified version of PTUMR is similar to the conventional version in terms of its fundamental grouping method, which divides the workers into groups under the way explained in Section 5.3.3. In addition, under the modified PTUMR, if r_i of the next worker is 1.5 times larger than the average r_i of all workers which have been already selected in the group, no worker does not further join the group and the master will create another group by including the worker and its subsequent ones. I refer to this modified method as PUTMR with resource capacity threshold.

I evaluate the performance of PTUMR with resource capacity threshold in heterogeneous distributed computing environments where all resource capacities of each worker are randomly chosen according to the uniform distribution. Figures Figs. 5.13 and 5.14 shows the average normalized turnaround time T_{real}/T_{bound} and the maximum one for 100 trials as a function of the heterogeneity *het* of resource capacities. PTUMR with resource capacity threshold can hold the increase of the normalized turnaround time and achieve the application turnaround time close to the lower bound even in the worst case for a wide range of the heterogeneity *het*, while both the conventional PTUMR and UMR remarkably degrade their performance. From these results, I can say that how the workers are grouped strongly affects the application turnaround time, in particular, when the resource capacity is highly heterogeneous. Furthermore, PTUMR can achieve the excellent turnaround time performance by grouping the workers in an appropriate way even in the highly heterogeneous case.

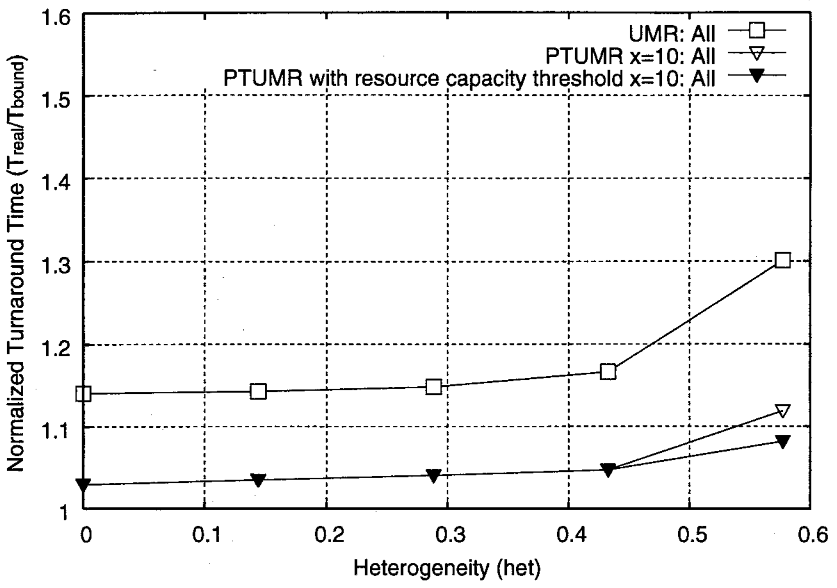

Figure 5.13: Impact of heterogeneity of resource capacities on average normalized turnaround time

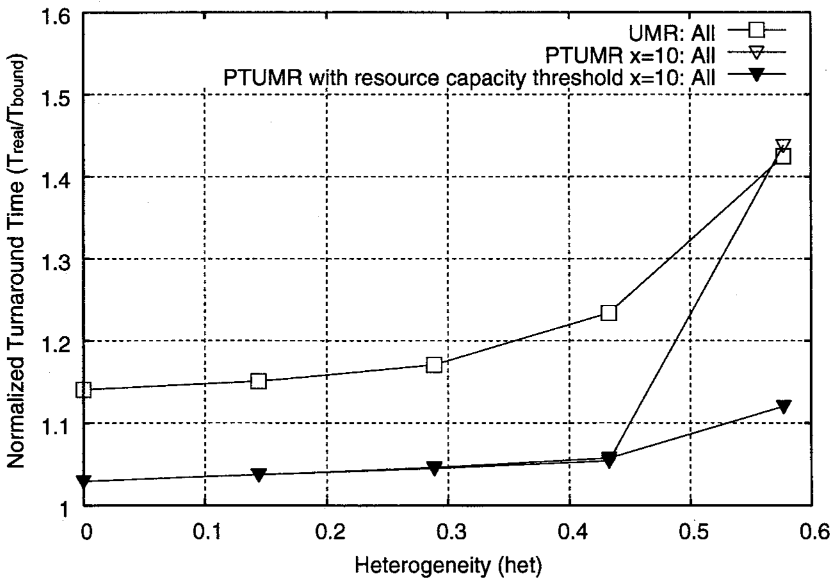

Figure 5.14: Impact of heterogeneity of resource capacities on maximum normalized turnaround time

5.5 Conclusion

The amount of data handled by distributed applications has recently increased, with the result that the time required for data transmission from the master to the workers has begun to degrade the application turnaround time. The adverse effects of data transmission time on the application turnaround time can be mitigated to a certain extent by employing a multiple-round scheduling algorithm such as UMR to overlap the data transmission time with the computation time. However, as UMR adopts the sequential transmission model, it cannot minimize the application turnaround time especially in asymmetric networks where the master-side link capacity is greater than the worker side link capacity.

In this chapter, a new multiple-round scheduling algorithm adopting a multiple transmission model was introduced as an extension of UMR that allows for application data to be transmitted to multiple workers in parallel by establishing multiple TCP connections simultaneously. The newly proposed algorithm PTUMR focuses on how to group the workers for parallel data transmissions, and determines appropriate parameters of the chunk size and the number of rounds that are nearly optimal in terms of achievable application turnaround time. Consequently, it can outperform the UMR for any application data size and network condition in various heterogeneous environment. Furthermore, to cope with the cases of considerably high heterogeneity, I improved the PTUMR to keep the heterogeneity of workers' capacities in each group within a certain degree, and showed that the improved version of PTUMR always achieved the application turnaround time close to the lower bound even in the distributed computing environments with high heterogeneity.

Appendix A: Derivation of Application Turnaround Time T_{real}

Derivation process of the application turnaround time T_{real} under PTUMR is illustrated in Fig. 5.15. In this figure, $comp_{j,k}$ and $comm_{j,k}$ is defined as the time when the virtual worker k completes processing chunks received in Round j, and when the master finishes transmitting the chunk to the virtual worker k in Round j, respectively.

The derivation process of T_{real} consists of three phases. In the first phase, I recursively derive the time at which each virtual worker completes processing chunks received in each round. This is shown in Fig. 5.15, line 6–14. Except for the last round (Round M-1), the size $C_{j,k}$ of the chunk transmitted to the virtual worker k in Round j corresponding to R_0 is derived according to Eqs. (5.10) and (5.12). In the second phase, the time when each virtual worker completes processing the last chunks is given on line 16–22. Note that the size of the last chunk, which is denoted by $C_{last,k}$, for the virtual worker k, should be chosen in a way that every virtual worker completes the processing of its last chunk with one accord, thereby preventing any virtual worker from entering the idle state before the entire workload has been finally processed. I call it the last-chunk alignment. The need for the last-chunk alignment arises from the difference in time at which each virtual worker began to process its initial chunk. Finally, T_{real} can be obtained on line 24.

Appendix B: Derivation of Application Turnaround Time T_{ideal} under Ideal Assumption

First I derive T'_{ideal}, which is the ideal application turnaround time in case without the last-chunk alignment (See Appendix A), as follows.

$$T'_{ideal} = \sum_{k=1}^{vN} \left(\frac{C_{0,k}}{vB_k} + v\epsilon_k \right) + \sum_{j=0}^{M-1} \left(\frac{C_{j,vN}}{vS_{vN}} + v\delta_{vN} \right). \tag{5.19}$$

```
1.  $comm_{-1,vN} = 0;$
2.  FOR $(k = 1;\ k \leq vN;\ k++)$
3.      $comp_{-1,k} = 0;$
4.  }
5.
6.  FOR $(j = 0;\ j < M - 1;\ j++)\{$
7.      $comm_{j,1} = comm_{j-1,vN} + \left(v\epsilon_1 + \frac{C_{j,1}}{vB_1}\right);$
8.      FOR $(k = 2; k \leq vN; k++)\{$
9.          $comm_{j,k} = comm_{j,k-1} + \left(v\epsilon_k + \frac{C_{j,k}}{vB_k}\right);$
10.     }
11.     FOR $(k = 1;\ k \leq vN;\ k++)\{$
12.         $comp_{j,k} = \max\{comp_{j-1,k}, comm_{j,k}\} + \left(v\delta_k + \frac{C_{j,k}}{vS_k}\right);$
13.     }
14. }
15.
16. $comm_{M-1,1} = comm_{M-2,vN} + \left(v\epsilon_1 + \frac{C_{last,1}}{vB_1}\right);$
17. FOR $(k = 2;\ k \leq vN;\ k++)\{$
18.     $comm_{M-1,k} = comm_{M-1,k-1} + \left(v\epsilon_k + \frac{C_{last,k}}{vB_k}\right);$
19. }
20. FOR $(k = 1;\ k \leq vN;\ k++)\{$
21.     $comp_{M-1,k} = \max\{comp_{M-2,k}, comm_{M-1,k}\} + \left(v\delta_k + \frac{C_{last,k}}{vS_k}\right);$
22. }
23.
24. $T_{real} = \max_{k}\{comp_{M-1,k}\};$
```

Figure 5.15: Derivation Process of Application Turnaround Time T_{real}.

which indicates the time at which the virtual worker vN completes processing the chunk received in the last round.

Next I consider the last-chunk alignment to reduce the application turnaround time. In order to have all virtual workers to complete processing the last round chunk with one accord, the master modifies the size of chunks allocated to virtual workers in the last round. Here, the difference Δ_k in the time required for the virtual worker k to perform computation of the last chunk and that for the virtual worker vN is expressed as follows.

$$\Delta_k = \sum_{K=k+1}^{vN} D_k = \left(\frac{C_{last,k}}{vS_k} + v\delta_k\right) - \left(\frac{C_{last,vN}}{vS_{vN}} + v\delta_{vN}\right). \tag{5.20}$$

where D_k indicates the difference in time at which the virtual worker $k-1$ start computing the last chunk and the time at which the virtual worker k start computing the last chunk. In my method, it is assumed that the amount of all chunks received by all virtual workers in the last round in case with the last-chunk alignment is equal to that in case without the last-chunk alignment. For this assumption, from Eq. (5.20), the relation between the total size R_{M-1} of the last chunks and the size $C_{last,vN}$ of the last chunk allocated to the virtual worker k can be formulated as follows.

$$C_{last,vN} = \frac{vS_{vN}}{\sum_{k=1}^{vN} vS_k}\left\{R_{M-1} + \sum_{k=1}^{vN} vS_k\left(v\delta_k - \sum_{K=k+1}^{vN} D_K\right)\right\} - vS_{vN} \times v\delta_{vN}. \tag{5.21}$$

In PTUMR, the difference in the time at which the master starts allocating the initial chunk to the virtual worker $k-1$ and that to the virtual worker k becomes D_k, because the computation time in each round is assumed to be identical for all virtual workers. Therefore, D_k is given by

$$D_k = \epsilon_k + \frac{C_{0,k}}{vB_k}. \tag{5.22}$$

Under the last-chunk alignment, the application turnaround time T_{ideal} is smaller than that T'_{ideal} in case without the alignment by the difference in the computation time of the chunk of $C_{M-1,vN}$ and that of $C_{last,vN}$. Therefore, from Eqs. (5.20), (5.21) and (5.22), T_{ideal} is formulated as follows.

$$\begin{aligned}
T_{ideal} &= T'_{ideal} - \left\{ \left(\frac{C_{M-1,vN}}{vS_{vN}} + v\delta_{vN} \right) - \left(\frac{C_{last,vN}}{vS_{vN}} + v\delta_{vN} \right) \right\}, \\
&= \frac{1}{\sum_{k=1}^{vN} vS_k} \left\{ W + M \times \sum_{k=1}^{vN} (vS_k \times v\delta_k) \right. \\
&\quad \left. + \sum_{k=1}^{vN} \left[vS_k \times \sum_{K=1}^{k} \left(\frac{\alpha_K \times \left(\frac{1-\theta}{1-\theta^M} \times (W - M\gamma) + \gamma \right) + \beta_K}{vB_K} + v\epsilon_K \right) \right] \right\}.
\end{aligned} \tag{5.23}$$

Chapter 6

Concluding Remarks

In this dissertation, in order to efficiently provide the next generation distributed applications such as the Grid computing and the P2P application, I have focused on the extension/modification of the existing resource management architecture which helps the users to discover the required resource and the existing scheduling algorithm which efficiently utilizes the gathered resource for the users' objective.

First, in Chapter 2, I have presented the resource management architecture for the Grid computing and P2P application. Especially, for the P2P network, I have shown the replication method which improves the search performance and the performance in load-balancing by distributing the replicas of the original data over the entire P2P network. In addition, the scheduling strategies for the Grid computing have been shown.

Next, in Chapter 3, I have focus on the resource management architecture for the P2P application, and have presented new replication methods balancing the storage load between peers in the P2P network. Due to the characteristic of the P2P network that the degree of each peer follows the power-law, the storage load concentrates on a small number of high-degree peers. In order to achieve the storage load-balancing without deteriorating the search performance too much, I have proposed two replication methods which make replicas of the data on some chosen peers. One method, *Path Random Replication*, chooses the peers randomly with a predetermined replication ratio, and the another method, *Path Adaptive Replication*, further improves the procedure in the decision to make a replica on a peer depends on how much storage is still available on it as well as the predetermined replication ratio. I have evaluated the performances of both methods through computer simulations. My results have shown that the latter method, *Path Adaptive Replication*, could extremely improve the performance in load balancing by using each peer's local information on resource availability. Furthermore, I considered the case where new data is added to the system; in order for the system to work well, I have modified *Path Adaptive Replication* by considering when the required data is allocated on the peer as well as considering the information on resource availability. As a result, *Path Adaptive Replication* have achieved effective load balancing while limiting the degradation of the search performance within an acceptable level.

In Chapter 4, in order to efficiently utilize the resources, I have evaluated the impact of the characteristics of the resource management architecture for the Grid environment on the resource selection for application tasks. First I have analytically evaluated performances of the task allocation schemes by modeling each computer as an M/G/1-PS queue. I have adopted three task allocation schemes: random selection of computers (Scheme 1), selection of least-loaded computers based on the current resource availability (Scheme 2), and random selection of computers from among them with processing capability greater than a predefined threshold (Scheme 3). Through numerical results, I have found that Scheme 2 outperforms the other schemes in terms of the task execution time, and selects a few computers with the highest performance. On the other hand, Scheme 3 achieves moderate performance over a wide range of average CPU utilization without concentrating task allocation on specific computers. Next, I have evaluated the impact of the concentration of task allocation through the computer simulation with respect to the realistic Grid environment, where information related to the resource utilization in each computer is updated periodically and all computers act as "users" which allocate application tasks each other. As a result, some randomness in Scheme 3 have succeeded in handling the uncertainty imposed by both the dynamic change of resource utilization and the periodic update of resource information, whereas Scheme 2 leads to critical performance degradation due to the concentration of task allocation with a long update period.

In Chapter 5, by focusing on the scheduling algorithm for the Grid computing which efficiently utilize the resources gathered for the application processing, I have proposed a new multi-round scheduling algorithm. The existing algorithm, UMR, can mitigate the adverse effect of the transmission time of a large amount of application data by overlapping the data transmission time and the computation time. However, UMR adopts the sequential transmission model, it cannot minimize the application turnaround time especially in asymmetric networks where the master-side link capacity is greater than the worker side link capacity. Therefore, I have proposed a new multiple-round scheduling algorithm, PTUMR, adopting a multiple transmission model which allows the master to transmit the application data to multiple workers in parallel. Consequently, The PTUMR outperforms the UMR for any application data size and any network condition in various heterogeneous environments. Furthermore, in order to cope with the cases of considerably high heterogeneity, I have modified the PTUMR to keep the heterogeneity of workers' capacities in each group within a certain degree, and showed that the improved version of PTUMR always achieved the application turnaround time close to the lower bound even when the distributed computing environments have high heterogeneity.

In this dissertation, I have focused on the resource management architecture and resource scheduling algorithm in the static environment where both the arrival and departure of end-computers do not occur. I will propose a new resource management architecture which handles the dynamic distributed environment and which always achieve the high-performance in terms of the lookup of resources and load-balancing. Furthermore, I will build the resource management architecture which parses the characteristics and requirements of the scheduling algorithm and which discovers the resource required for the algorithm so as to achieve the high-performance application processing.

Bibliography

[AB00] R. Albert and A. L. Barabasi, "Topology of Evolving Networks: Local Events and Universality", *Physical Review Letters*, Vol. 85, No. 24, pp. 5234–5237, December 2000.

[AH99] L. A. Adamic and B. A. Huberman, "The nature of markets in the world wide web", in *Proc. of the Fifth International Conference of the Society for Computational Economics, Computing in Economics and Finance*, June 1999.

[ALPH01] L. A. Adamic, R. M. Lukose, A. R. Puniyami, and B. A. Huberman, "Search in power-law networks", *Physical Review E*, Vol. 64, 046135, No. 4, October 2001.

[AMG03] S. Ata, M. Murata, and Y. Gotoh, "Replication Strategies in Peer-to-Peer Services over Power-law Overlay Networks", in *Proc. of Asia-Pacific Network Operations and Management (APNOMS2003)*, pp. 84–95, October 2003.

[BA99] A. L. Barabasi and R. Albert, "Emergence of Scaling in Random Networks", *SCIENCE*, Vol. 286, No. 5439, pp. 509–512, October 1999.

[BCF$^+$02] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y Rebert, "Bandwidth-centric allocation of independent tasks on heterogeneous platforms", in *Proc. of the international Parallel and Distributed Processing Symposium (IPDPS'02)*, April 2002.

[BDET00] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", in *Proc. of International Conference Measurement and Modeling of Computer Systems (SIGMetrics 2000)*, pp. 34–43, June 2000.

[BGMR96] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press, 1996.

[Bha05] K. Bhatia, "Peer-to-Peer Requirements On The Open Grid Services Architecture Framework", Grid Forum Working Draft GFD-I.049, July 2005, URL: ⟨http://www.ggf.org⟩.

[BLR03] O. Beaumont, A. Legrand, and Y. Robert, "Optimal Algorithms for Scheduling Divisible Workloads on Heterogeneous Systems", in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, April 2003.

[BT02] T. Bu and D. Towsley, "On Distinguishing between Internet Power Law Topology Generators", in *Proc. of IEEE INFOCOM 2002*, pp. 638–647, June 2002.

[CBLR02] C. Cyril, O. Beaumont, A. Legrand, and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Grids", Technical Report 2002-12, LIP, March 2002.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing", in *Proc. of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC–10)*, August 2001.

[CFK+00] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Network and Computer Applications*, Vol. 23, No. 3, pp. 187–200, July 2000.

[CMS04] C. Christos, M. Mihail, and A. Saberi, “Random Walks in Peer-to-Peer Networks”, in *Proc. of IEEE INFOCOM 2004*, March 2004.

[CRB$^+$03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making Gnutella-like Systems Scalable”, in *Proc. of ACM SIGCOMM 2003*, August 2003.

[CS02] E. Cohen and S. Shenker, “Replication Strategies in P2P Networks”, in *Proc. of ACM SIGCOMM 2002*, August 2002.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A Distributed Anonymous Information Storage and Retrieval System”, in *Proc. of ICSI Workshop on Design Issue in Anonymity and Unobservability, LNCS 2009*, pp. 46–66, July 2000.

[FK98a] I. Foster and C Kesselman, *The GRID Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.

[FK98b] I. Foster and C. Kesselman, “The Globus Project: A Status Report”, in *Proc. of IPPS/SPDP ’98 Heterogeneous Computing Workshop*, pp. 4–18, March 1998.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the grid: Enabling scalable virtual organizations”, *International Journal of Supercomputer Applications*, Vol. 15, No. 3, pp. 200–222, 2001.

[Fos05] I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems”, in *Proc. of International Conference on Network and Parallel Computing, LNCS 3779*, pp. 2–13, December 2005.

[GCKF01] S. Gullapalli, K. Czajkowski, C. Kesselman, and S. Fitzgerald, “Grid Notification Framework (VB0.1)”, Grid Forum Working Draft GWD-GIS-019, June 2001, URL: ⟨http://www.gridforum.org⟩.

[GGF] "The global grid forum (ggf)", URL: ⟨http://www.ggf.org/⟩.

[Glo] "The globus alliance", URL: ⟨http://www.globus.org/⟩.

[GLS+04] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems", in *Proc. of INFOCOM 2004*, March 2004.

[Gnu] "Gnutella.com", URL: ⟨http://www.gnutella.com/⟩.

[GO93] D. Gerogiannis and S. C. Orphanoudakis, "Load Balancing Requirements in Parallel Implementations of Image Feature Extraction Tasks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 9, pp. 994–1013, 1993.

[GS05] P. B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Table", in *Proc. of INFOCOM 2005*, March 2005.

[GSBK04] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive Replication in Peer-to-Peer Systems", in *Proc. of 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pp. 360–369, March 2004.

[HSSY00] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing", in *Proc. of GRID 2000, LNCS 1971, Springer*, pp. 191–202, December 2000.

[Kle79] L. Kleinrock, *Queueing Systems Volume 2: Computer Applications*, John Wiley & Sons, 1979.

[KLS02] P. Keyani, B. Larson, and M. Senthil, "Peer Pressure: Distributed Recovery from Attacks in Peer-to-Peer Systems", in *Proc. of IFIP Workshop on Peer-to-Peer Computing, in conjunction with Networking*, pp. 306–320, May 2002.

[KR04] D. R. Kargar and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems", in *Proc. of the Sixteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2004)*, June 2004.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", in *Proc. of 16th ACM International Conference on Supercomputing (ICS'02)*, pp. 84–95, June 2002.

[LRS02] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable", in *Proc. of 1st International Workshop on Peer-to-Peer Systems*, pp. 94–103, March 2002.

[LW03] T. Lin and H. Wang, "Search Performance Analysis in Peer-to-Peer Networks", in *Proc. of the Third International Conference on Peer-to-Peer (P2P'03)*, pp. 204–205, September 2003.

[MBR03] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing In A Small World", in *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, March 2003.

[MDS] "GT Information Services: Monitoring & Discovery System (MDS)", URL: ⟨http://www.globus.org/toolkit/mds/⟩.

[MYCR05] L. Marchal, Y. Yang, H. Casanova, and Y Robert, "A realistic network/application model for scheduling divisible loads on large-scale platforms", in *Proc. of the international Parallel and Distributed Processing Symposium (IPDPS'05)*, April 2005.

[Nap] "Napster", URL: ⟨http://www.napster.com/⟩.

[New05] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law", Technical Report of Department of Physics and Center for the Study of Complex Systems, University of Michigan, MI 48109, January 2005.

[RD01a] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", in *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, November 2001.

[RD01b] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", in *Proc. of 18th Symposium on Operating Systems Principles*, pp. 188–201, October 2001.

[RFHK01] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A Scalable Content-Addressable Network", in *Proc. of ACM SIGCOMM 2001*, August 2001.

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT", in *Proc. of the USENIX Annual Technical Conference*, June 2004.

[Rob03] T. G. Robertazzi, "Ten Reasons to Use Divisible Load Theory", *IEEE Computer*, Vol. 36, No. 5, pp. 63–68, May 2003.

[Ros01] A. L. Rosenberg, "Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier Is Not Better", in *Proc. of the 3rd IEEE International Conference on Cluster Computing (Cluster 2001)*, pp. 124–131, October 2001.

[SAZ$^+$04] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet Indirection Infrastructure", *IEEE/ACM Transactions on Networking*, Vol. 12, No. 2, pp. 205–218, April 2004.

[SGG02] S. Sarou, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", Technical Report of Department of Computer Science and Engineering, University of Washington, UW-CSE-01-06-02, 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", in *Proc. of ACM SIGCOMM 2001*, August 2001.

[STI02] H. Sunaga, M. Takemoto, and T. Iwata, "Advanced Peer-to-Peer Network Platform for Various Services – SIONet (Semantic Information Oriented Network) –", in *Proc. of the Second International Conference on Peer-to-Peer (P2P'02)*, pp. 169–170, September 2002.

[WSH99] R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Future Generation Computing Systems*, Vol. 15, No. 5–6, pp. 757–768, October 1999.

[YC02] Y. Yang and H. Casanova, "A Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation", Technical Report of Department of Computer Science and Engineering, University of California CS20020721, 2002.

[YC03a] Y. Yang and H. Casanova, "RUMR: Robust Scheduling for Divisible Workload", in *Proc. of the Twelves IEEE International Symposium on High-Performance Distributed Computing (HPDC–12)*, June 2003.

[YC03b] Y. Yang and H. Casanova, "UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads", in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, April 2003.

[YKTO03] H. Yamamoto, K. Kawahara, T. Takine, and Y. Oie, "Performance Comparison of Process Allocation Schemes Depending upon Resource Availability on Grid Computing Environment", in *Proc. of International Conference on Computational Science, LNCS 2658*, Vol. 2, pp. 247–256, June 2003.

[YKTO06] H. Yamamoto, K. Kawahara, T. Takine, and Y. Oie, "Performance Comparison of Process Allocation Schemes Depending upon Resource Availability on Grid Computing Environment", *IEICE Transactions on Information and Systems – Special Section on Parallel/Distributed Computing and Networking*, Vol. E89-D, No. 2, pp. 459–468, February 2006.

[YMO05] H. Yamamoto, D. Maruta, and Y. Oie, "Replication Methods for Load Balancing on Distributed Storages in P2P Networks", in *Proc. of the 2005 International Symposium on Applications and the Internet (SAINT2005)*, pp. 264–271, January 2005.

[YMO06] H. Yamamoto, D. Maruta, and Y. Oie, "Replication Methods for Load Balancing on Distributed Storages in P2P Networks", *IEICE Transactions on Information and Systems – Special Section on New Technologies and their Applications of the Internet III*, Vol. E89-D, No. 1, pp. 171–180, January 2006.

[YTO05] H. Yamamoto, M. Tsuru, and Y. Oie, "Parallel Transferable Uniform Multi-Round Algorithm for Achieving Minimum Application Turnaround Times for Divisible Workload", in *Proc. of the 2005 International Conference on High Performance Computing and Communications (HPCC-05), LNCS 3726*, pp. 817–828, September 2005.

[ZHS+04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment", *IEEE Journal on Selected Area in Communications*, Vol. 22, No. 1, pp. 41–53, January 2004.