Thesis for the Degree of
Doctor of Philosophy

Group-based Adaptive Scheduling Mechanism in Desktop Grid

by

SungJin Choi

Department of
Computer Science and Engineering

Graduate School

Korea University

June 2007

Abstract

Desktop Grid has recently been an attractive computing paradigm for high throughput applications. However, Desktop Grid computing is complicated by heterogeneous capabilities, failures, volatility, and lack of trust because it is based on desktop computers at the edge of the Internet. In a Desktop Grid computing environment, volunteers (that is, resource providers) have heterogeneous properties such as CPU, memory, network, etc. They are exposed to failures such as crash and link failures. In addition, volunteers can freely participate in the computations and dynamically leave them in the middle of execution. Moreover, some malicious volunteers may tamper with the computation and return corrupted results. These distinct features make it difficult for a Desktop Grid scheduler to allocate tasks to volunteers. Therefore, it is necessary to develop scheduling mechanisms that adapt to such a dynamic computing environment.

To explore the solution space, the thesis first provides comprehensive taxonomy and survey of Desktop Grid. In addition, it presents comprehensive taxonomy and survey of scheduling for Desktop Grid in order to better design and develop a new scheduling mechanism.

As the result of the taxonomy and survey, existing scheduling mechanisms did not adapt to a dynamic computing environment. Particularly, they did not consider volunteer's properties such as volatility, availability, and credibility that strongly affect reliability and performance. Moreover, they did not provide scheduling mechanisms on a per group basis. In other words, they did not apply different scheduling algorithms to each group according to volunteer's properties. As a result, they deteriorate the reliability of computation as well as performance.

To solve these problems, the thesis proposes a new group-based adaptive scheduling mechanism, which adapts to a dynamic Desktop Grid computing environment. The group-based adaptive scheduling mechanism classifies and constructs groups according to volunteer's properties such as dedication, volatility, availability, and credibility. Then it applies different scheduling, replication, result certification, and fault tolerance algorithms to each group. Consequently, it improves reliability and performance. The simulation results show that how it can outperform existing scheduling mechanisms.

Contents

1 Introduction **1**

1.1 Desktop Grid Computing 1

1.2 Motivation 3

1.2.1 Why Group-based Adaptive Scheduling? 4

1.3 Contribution 9

1.4 Thesis Organization 10

2 Desktop Grid: Overview and Taxonomy **12**

2.1 Overview 13

2.2 Desktop Grid vs. Grid 15

2.3 Taxonomy of Desktop Grid 17

2.3.1 Organization 18

2.3.2 Platform 23

2.3.3 Scale 24

2.3.4 Resource Provider 24

2.4 Taxonomy of Desktop Grid Scheduling 25

2.4.1 Application's Perspective Considerations 25

2.4.2 Resource's Perspective Considerations 27

2.4.3 Scheduler's Perspective Considerations 31
2.5 Survey of Desktop Grid Systems 41
2.5.1 BOINC 42
2.5.2 XtremWeb 50
2.5.3 Entropia 51
2.5.4 Bayanihan 52
2.5.5 Javelin 54
2.5.6 CPM 55
2.5.7 Charlotte 57
2.5.8 POPCORN 57
2.5.9 WebCom 58
2.5.10 CCOF 59
2.5.11 Organic Grid 61
2.5.12 Messor 62
2.5.13 Paradropper 63
2.5.14 Condor 64
2.5.15 Kondo et al. 65
2.6 Discussion 66
2.6.1 Challenging Issues for Desktop Grid 66
2.6.2 A Direction for Desktop Grid Scheduling 69
2.7 Related Work 73
2.7.1 Taxonomy and Survey of Grid and Desktop Grid 73
2.7.2 Taxonomy and Survey of Scheduling 73

3 System Model **76**

3.1 Execution Model 76

3.2 Failure Model 78

4 Group-based Adaptive Scheduling Mechanism **83**

4.1 Resource Grouping Method 83

4.1.1 Criteria for Resource Grouping 84

4.1.2 Constructing and Characterizing Volunteer Groups 89

4.1.3 Maintaining Volunteer Groups 96

4.2 Agent-based Group Scheduling 97

4.2.1 Why Agent? 97

4.2.2 Agent based Desktop Grid Computing Model . . 99

4.2.3 Allocating Scheduling Agents to Scheduling Groups 102

4.2.4 Distributing Task Agents to Group Members . . . 104

4.3 Group Scheduling for Replication 106

4.3.1 How to Calculate the Number of Redundancy . . 107

4.3.2 How to Select Replicas 111

4.3.3 How to Distribute Tasks to Replicas 112

4.4 Group Scheduling for Result Certification 115

4.4.1 Applying Result Certification to Volunteer Group 115

4.4.2 Scheduling Algorithm for Result Certification . . 116

4.5 Fault Tolerant Algorithm 119

4.5.1 Handling Failure of Scheduling Agent 121

4.5.2 Handling Failure of Task Agent 123

4.6 Related Work 127

4.6.1 Scheduling and Fault tolerance 127

4.6.2 Scheduling for Replication 131

4.6.3 Scheduling for Result Certification 133

5 Performance Evaluation 135

5.1 Group-based Adaptive Scheduling 135

5.2 Group Scheduling for Replication 147

5.3 Group Scheduling for Result Certification 151

6 Conclusion 158

6.1 Contributions 158

6.2 Future Work 161

Bibliography 163

Acknowledgements 181

List of Figures

2.1 Desktop Grid computing environment 14

2.2 A taxonomy of Desktop Grid 17

2.3 Execution model of centralized Desktop Grid 19

2.4 Execution model of distributed Desktop Grid 22

2.5 Application's perspective considerations 26

2.6 Resource's perspective considerations 28

2.7 Scheduler's perspective considerations 32

2.8 Scheduler's perspective considerations (continued) . . . 33

3.1 New execution model of Desktop Grid 77

4.1 Criteria for resource grouping 85

4.2 Volunteer groups according to α_v and Θ 91

4.3 Algorithm of volunteer group construction according to α_v and Θ 92

4.4 Volunteer groups according to α_v, Θ, and C_v 94

4.5 Algorithm of volunteer group construction according to α_v, Θ, and C_v 95

4.6 Agent based Desktop Grid computing model 100

4.7 Algorithm of deputy volunteer selection 104
4.8 Algorithm for calculating the number of redundancy . . . 110
4.9 Parallel and sequential distribution 113
4.10 Fault tolerant algorithm in the presence of failures of S-MA 122
4.11 Fault tolerant algorithm in the presence of failures of T-MA (1) 124
4.12 Fault tolerant algorithm in the presence of failures of T-MA (2) 125
4.13 Fault tolerant algorithm in the presence of failures of T-MA (3) 126

5.1 The average number of completed tasks 139
5.2 The average number of completed tasks in the case of replication in Case 2 141
5.3 The average number of redundancy in Case 2 142
5.4 The average number of completed tasks in case of replication (reliability threshold = 0.8) 144
5.5 The average number of redundancy 145
5.6 The average number of redundancy (Continued) 146
5.7 The average number of redundancy with majority voting 150
5.8 The number of completed tasks with majority voting . . 151
5.9 Total number of task without result certification 154
5.10 Total number of task with result certification 155
5.11 Result certification: majority voting 156

5.12 Result certification: spot-checking 157

List of Tables

2.1 A comparison of Grid and Desktop Grid 16

2.2 A Survey of Desktop Grid 42

2.3 Survey of Desktop Grid systems focusing on scheduling . 43

2.4 Survey of Desktop Grid systems focusing on scheduling (Continued) 44

2.5 Survey of Desktop Grid systems focusing on scheduling (Continued) 45

2.6 Survey of Desktop Grid systems focusing on scheduling (Continued) 46

2.7 Survey of Desktop Grid systems focusing on scheduling (Continued) 47

2.8 Survey of Desktop Grid systems focusing on scheduling (Continued) 48

3.1 Notations for volunteer autonomy failures 80

4.1 The combination of volunteer groups 103

5.1 Simulation environment 138

5.2 Simulation environment for replication 148
5.3 Simulation environment for result certification 152

Chapter 1

Introduction

1.1 Desktop Grid Computing

Grid has recently emerged as a promising paradigm for high performance or high throughput computing because of the vast development of powerful computers and high-speed network technologies as well as low cost [1, 2, 3, 4, 5]. Grid aims to aggregate heterogeneous, large-scale, and multiple-institutional resources, and to provide the transparent, secure, and coordinated access to various computing resources (supercomputer, cluster, scientific instruments, database, storage, etc.) owned by multiple institutions by making virtual organization [1, 2, 3, 4, 5]. On the other hand, Desktop Grid[1] aims to harvest a number of idle desktop computers owned by individuals at the edge of Internet [5, 13, 17, 21, 24, 50, 53]. Desktop Grid has recently received the rapidly growing

[1]Desktop Grid computing is also called volunteer computing [13, 24], global computing [18, 26, 43, 47], Peer-to-Peer Grid computing [36, 54], public-resource computing [14], and Peer-to-Peer cycle sharing systems [37].

interest and attraction because of the success of the most popular examples such as GIMPS [10], distributed.net [11], and SETI@Home [12]. Some studies have been made on Desktop Grid systems which provide an underlying platform: BOINC [13, 14, 16], XtremWeb [17, 18, 19], Entropia [20, 21], Bayanihan [22, 23, 24], Javelin [25, 26, 27], Computer Power Market (CPM) [28, 29], Charlotte [30], POPCORN [31], WebCom [32, 33], Cluster Computing On the Fly(CCOF) [34, 35], Organic Grid [38, 39], Messor [40, 41], Paradropper [42, 43], Condor [44, 45, 46], Korea@Home [52], and so on.

A Desktop Grid computing environment mainly consists of client, volunteer, and server. A client is a parallel job submitter. A volunteer is a resource provider that donates its computing resources when idle. A server is a central manager that controls submitted jobs and volunteers. A client submits a parallel job to a server. A job is divided into sub-jobs that have their own specific input data. The sub-job is called a task. The server allocates tasks to volunteers using scheduling mechanisms. Each volunteer executes its task when idle, while continuously requesting data from its server. When each volunteer subsequently finishes its task, it returns the result of the task to the server. Finally, when the server collects all results of tasks from volunteers, it returns the final result of the job back to the client.

1.2 Motivation

Desktop Grid is a kind of Grid, but there are distinct several differences between them in terms of the types and characteristics of resources, and the types of sharing (see Table 2.1 in Chapter 2). In particular, Desktop Grid computing is complicated by heterogeneous capabilities, failures, volatility (that is, intermittent presence), and lack of trust [6, 7, 8, 9, 13, 14, 15, 18, 20, 23, 36, 37, 47, 48, 53, 54] because it is based on desktop computers (or volunteers) at the edge of the Internet.

Volunteers have heterogeneous capabilities (that is, CPU, memory, network bandwidth, and latency), and are exposed to link and crash failures. In particular, they are voluntary participants that do not receive any reward for donating their resources. As a result, they are free to join and leave in the middle of execution without any constraints. Accordingly, they have heterogeneous volunteering times (that is, the time of donation), and *public execution* (that is, the execution of a task as a volunteer) can be stopped arbitrarily on account of unexpected leave. Moreover, public execution is temporarily suspended by *private execution* (that is, the execution of a private job as a personal user) because volunteers are not totally dedicated to public executions. In this thesis, these unstable situations are regarded as *volunteer autonomy failures* because they lead to the delay and blocking of the execution of tasks and include situations resulting in the partial or entire loss of the executions. Volunteers have different occurrence rates for *volunteer* *autonomy failures* according to their execution behavior. In addition, some malicious volunteers may tamper with the computation and return corrupt results. A variety of hardware and software lead to deviation from the result of a task. These distinct features make it difficult for a server to schedule tasks and manage allocated tasks and volunteers. Therefore, it is necessary to develop scheduling mechanisms that adapt to such a dynamic computing environment.

1.2.1 Why Group-based Adaptive Scheduling?

Scheduling is fundamentally important to develop a Grid system. Scheduling is the process of assigning tasks to the most suitable resource providers (that is, where to execute tasks) and ordering tasks (that is, when to execute a task) [60, 61, 62, 66, 67, 73]. In order to decide where to execute a task in Grid, information gathering about the resources, resource discovery that looks for available and potential resources, resource selection, and monitoring of task execution are involved because of the heterogeneous and dynamic nature of Grid resources. Ordering tasks means placing priority on tasks to be executed at a specific node or site. Grid generally performs scheduling in a hierarchical manner [69, 72, 75]. In order words, Grid scheduler consists of meta scheduler (or super scheduler) and local scheduler. Generally, a meta scheduler is responsible for where to execute tasks among multiple sites, whereas a local scheduler is responsible for assigning and ordering tasks within one site [60, 69, 72, 75]. LoadLeveler, LSF, or PBS can be used as a local scheduler [60, 69, 72, 75].

Scheduling of Desktop Grid is different from that of Grid because Desktop Grid is different from Grid in terms of the type of resource, dedication, trust, failure, application, and so on [5, 14, 21, 24, 48, 53, 54] (see Table 2.1, Chapter 2). First, Desktop Grid scheduling is mainly the process of assigning tasks to the most suitable resources (that is, to decide where to execute tasks) [16, 19, 21, 24, 27, 30, 36, 38, 41, 49, 50, 54]. It is performed in a centralized way or in a fully distributed way. Second, most Desktop Grid systems do not need a local scheduler like Grid in the sense that a scheduling target is a single desktop computer, not a site like Grid (that is, multiple processors or computers in supercomputer or cluster). Third, Desktop Grid scheduling is complicated by heterogeneous, volatile, faulty, and malicious resources. Desktop Grid scheduler focuses more on volatility (non-dedication), lack of trust, and heterogeneous properties than Grid scheduler [23, 36, 37, 47, 49, 53, 54, 55, 56, 57, 58, 80]. Finally, Desktop Grid scheduling is opportunistic. Desktop Grid respects the autonomy of volunteers (that is, volunteers can freely participate in public execution). Thus, Desktop Grid scheduling should use resources as quickly as possible when they are available or idle [45]. Therefore, it is necessary to develop a new Desktop Grid scheduler.

In order to consider these distinct features, a Desktop Grid scheduler needs a resource grouping method, which ensures that volunteers with similar properties (such as capability, performance, availability, workload, reputation/trust, volatility, etc.) are grouped together. It needs to apply scheduling algorithms to each group depending on group's characteristics. Coupling a resource grouping method with scheduling helps schedule and manage tasks efficiently. It is very important how to group volunteers depending on what properties, because scheduling and resource management are performed on the basis of characteristics of the groups. There are benefits by coupling a resource grouping method with scheduling as follows.

1) ***The coupling method enables a scheduler to apply various replication, result certification, and fault tolerant algorithms to each homogeneous group***. Resource grouping makes it possible to form homogeneous groups according to availability, volatility, and trust. A scheduler can apply various replication, result certification, and fault tolerant algorithms to each homogeneous group. In other words, the coupling method makes it possible that a scheduler selects and applies replication, result certification, and fault tolerant algorithms suitable for the characteristics of each group. For example, a scheduler can take many replicas for high-volatile groups. A scheduler can frequently check result correctness for distrusted groups. However, in existing Desktop Grid systems, resource grouping is not tightly related with scheduling (especially, result certification, replication, and reassignment). As a result, there are a lot of overhead and performance

degradation.

2) ***The coupling method easily enables reputation-based or incentive-based scheduling***. In Desktop Grid, resources can be eager, volatile, selfish, or malicious. In other words, resources have various volunteering time, volatility, credibility, and availability according to their behavior. In order to score and rank the resources, and then reward or punish them, a reputation system is necessary. Reputation-based scheduling couples scheduling with a reputation system. As a result, reputation-based scheduling can choose more high-quality resources, so that it can improve the reliability and performance. Incentive-based scheduling focuses on punishing (for example, exclusion) volatile, selfish, or malicious resources. Consequently, repuation/intensive-based scheduling encourages volunteers to donate their resources eagerly and reliably. Resource grouping makes it easier that a scheduler applies reputation or incentive to each group, in the sense that each group has distinct properties and characteristics (such as eager, reliable, malicious, volatile, and selfish).

3) ***The coupling method improves reliability and performance***. Volunteer groups can be formed according to volatility, volunteering time, credibility, and reputation/trust. Thus, the coupling method directly affects reliability, completion time, and result correctness. In addition, the coupling method reduces the overhead and latency by applying replication, result certification suitable for each group's properties. For example, a scheduler can take the large number of repli-

cas for unreliable group to improves reliability. It can take the small number of replicas for low-volatile groups to improves performance. It also can apply the small frequency of result certification for highly-reliable groups while satisfying correctness threshold. As a result, it improves reliability of computation and performance.

Although the coupling method has a lot of advantages, existing scheduling mechanisms, however, did not consider volunteer's properties such as volatility, availability, and credibility that strongly affect reliability, performance, and result correctness. Moreover, they did not provide scheduling mechanisms on a per group basis. In other words, they can not apply different scheduling algorithms to each group according to volunteer's properties. As a result, they deteriorate the reliability of computation as well as performance.

To solve these problems, the thesis proposes a new group-based adaptive scheduling mechanism, which adapts to a dynamic Desktop Grid computing environment. The group-based adaptive scheduling mechanism classifies and constructs groups according to volunteer's properties such as dedication, volatility, availability, and credibility. Then it applies different scheduling, replication, result certification, and fault tolerance algorithms to each group. Consequently, it improves reliability and performance. The simulation results show that how much it can outperform existing scheduling mechanisms.

1.3 Contribution

This thesis has several novel contributions towards improving the understanding of Desktop Grid and towards advancing the area of scheduling in Desktop Grid. The contributions of this thesis are as follows:

1. **Taxonomy and Survey of Desktop Grid**

   This thesis discusses the key concepts and characteristics about Desktop Grid. It defines architecture and execution model of Desktop Grid and classifies volunteer properties. Particularly, it classifies and defines volunteer autonomy failures conceptually. It also provides a new comprehensive taxonomy and survey of Desktop Grid. The taxonomy and survey help understand the definition, architecture, model, and applications of Desktop Grid.

2. **Taxonomy and survey of Desktop Grid Scheduling**

   This thesis presents the key functionalities that Desktop Grid scheduling must support. It provides comprehensive taxonomy and survey of Desktop Grid scheduling. The taxonomy and survey help model and develop a new scheduling mechanism.

3. **Resource Grouping Method**

   This thesis proposes a resource grouping method, which classifies and constructs groups according to volunteer's properties such as volatility, dedication, availability, and credibility.

4. **Group-based Adaptive Scheduling Mechanism**

   This thesis proposes a new group-based dynamic scheduling mechanism, which adapts to a dynamic Desktop Grid computing environment. The new scheduling mechanism couples resource grouping with scheduling. It applies various replication, result certification, and fault tolerant algorithms to each homogeneous group. This thesis presents agent-based scheduling, group-based scheduling for replication, group-based scheduling for result certification, and fault tolerant scheduling. It also provides the simulation results showing that how the group-based dynamic scheduling mechanism can outperform existing scheduling mechanisms.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

- Chapter 2 presents an overview and taxonomy of Desktop Grid. We classify and characterize Desktop Grid and scheduling. The proposed taxonomy is mapped to various Desktop Grid systems. From the taxonomy and survey, we extract the issues and challenges of Desktop Grid scheduling. Furthermore, we present a direction for Desktop Grid scheduling.

- Chapter 3 presents a system model defined in this thesis. We newly define a execution model of Desktop Grid, and volunteer autonomy failures that describe volatility and dedication of volunteers.

- Chapter 4 proposes a new group-based adaptive scheduling mechanism. It describes a resource grouping method. It also explains agent-based scheduling, group scheduling for replication, group scheduling for result certification, and fault tolerant scheduling algorithms.
- Chapter 5 shows the performance evaluation. We evaluate the group-based adaptive scheduling mechanism in terms of the number of the competed tasks, the number of redundancy, spot-checking rate, and error rate.
- Chapter 6 summarizes the results and impacts of this thesis and discusses various avenues of future work.

Chapter 2

Desktop Grid: Overview and Taxonomy

Desktop Grid has been recently studied, but there is no general taxonomy and survey for it. Therefore, it is difficult to understand the definition, architecture, model, and applications of Desktop Grid, as well as to design and develop Desktop Grid systems. Therefore, we propose a new comprehensive taxonomy and survey of Desktop Grid in order to characterize and categorize Desktop Grid.

In particular, scheduling of Desktop Grid is different from that of Grid because Desktop Grid is different from Grid in terms of the type of resource, dedication, trust, failure, application, and so on (see Table 2.1). Desktop Grid scheduling is considerably challenging because of unreliable, heterogeneous, volatile, and insecure environment. However, there is no taxonomy and survey of Desktop Grid scheduling. As a result, it is difficult to design and develop a new scheduling mechanism.

In this chapter, we characterize Desktop Grid. We also propose a new comprehensive taxonomy and survey of scheduling for Desktop Grid in order to help model and develop a new scheduling mechanisms. Then, we discuss a direction of Desktop Grid scheduling.

2.1 Overview

Desktop Grid aims to harvest a number of idle desktop computers owned by individuals at the edge of Internet [5, 13, 17, 21, 24, 50, 53]. Desktop Grid systems usually support embarrassingly parallel applications which consist of a lot of instances of the same computation with each own data [1, 2, 14, 18, 20, 22, 25, 30, 52]. The applications are usually involved with scientific problems which need large amounts of processing capacity over long periods of time.

A Desktop Grid computing environment mainly consists of client, volunteer, and server, as shown in Figure 2.1. A client is a parallel job submitter. A volunteer is a resource provider that donates its computing resources when idle. A server is a central manager that controls submitted jobs and volunteers. A client submits a parallel job to a server. The job is divided into sub-jobs that have their own specific input data. The sub-job is called a task. The server distributes tasks to volunteers using scheduling mechanisms. Each volunteer executes its task when idle. When each volunteer subsequently finishes its task, it returns the result of the task to the server. Finally, the server returns

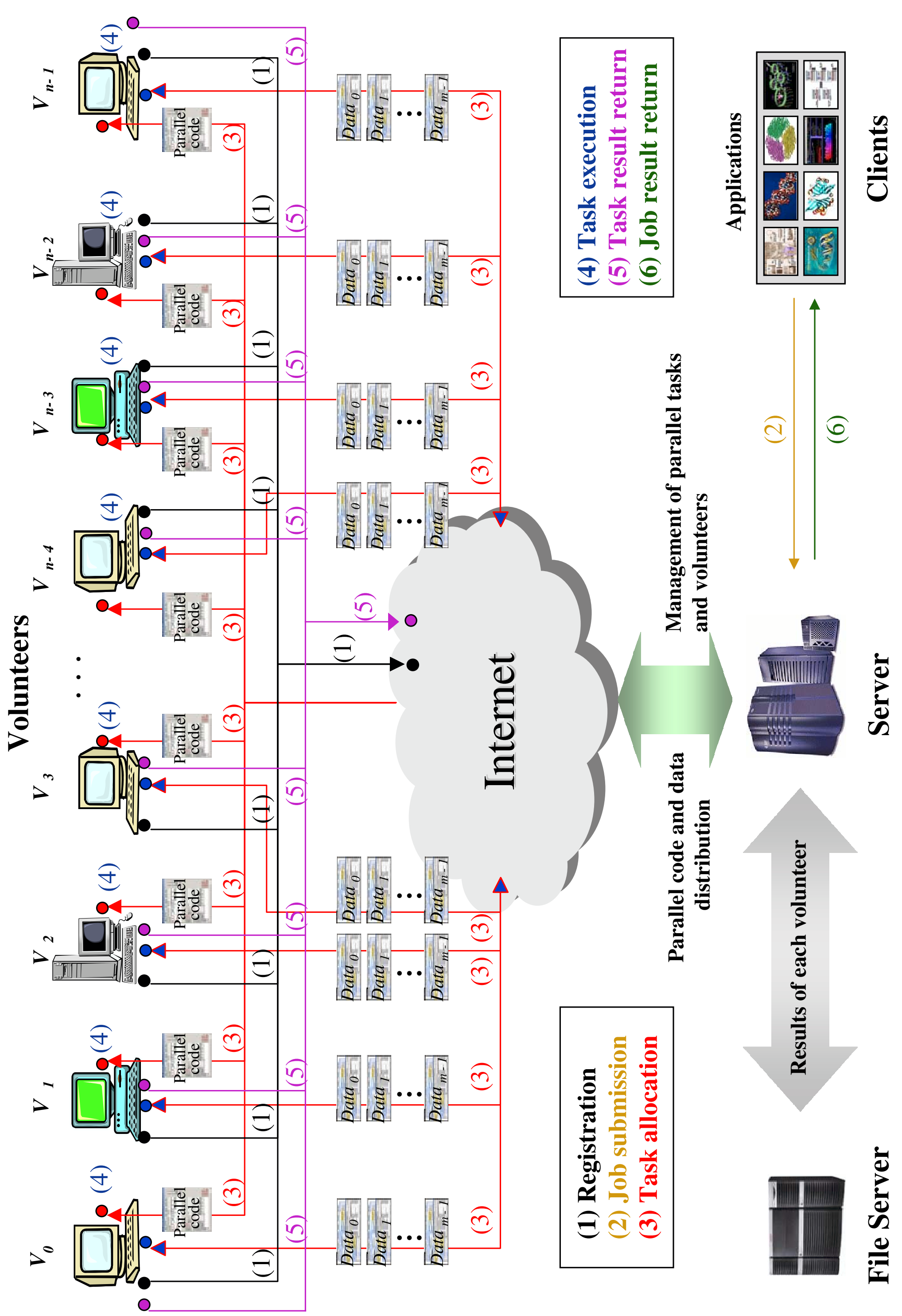


Figure 2.1: Desktop Grid computing environment

the final result of the job back to the client.

Desktop Grid has recently received the rapidly growing interest and attraction because of the success of the most popular examples such as GIMPS [10], distributed.net [11], and SETI@Home [12]. Some studies have been made on Desktop Grid systems which provide an underlying platform: BOINC [13, 14, 16], XtremWeb [17, 18, 19], Entropia [20, 21], Bayanihan [22, 23, 24], Javelin [25, 26, 27], Computer Power Market (CPM) [28, 29], Charlotte [30], POPCORN [31], WebCom [32, 33], Cluster Computing On the Fly(CCOF) [34, 35], Organic Grid [38, 39], Messor [40, 41], Paradropper [42, 43], Condor [44, 45, 46], Korea@Home [52, 53, 54, 55, 56, 57, 58], and so on.

2.2 Desktop Grid vs. Grid

Desktop Grid has recently received the strong attraction for executing high throughput applications as CPU, storage and network capacities improve and become cheaper. Desktop Grid is different from Grid in terms of the types and characteristics of resources, and the types of sharing [1, 2, 5, 6, 7, 9, 14, 21, 24, 53, 54] (see Table 2.1). The resources of Desktop Grid mainly are personal computers (that is, desktop), whereas Grid resources include supercomputer, cluster, scientific instruments, database, storage, etc. Desktop Grid resources are highly-volatile, non-dedicated, and highly-heterogeneous, differently from Grid resources. They also are more malicious, unreliable, and faulty than

Table 2.1: A comparison of Grid and Desktop Grid

Items	Desktop Grid (DG)		Grid
	Internet-based (Volunteer DG)	LAN-based (Enterprise DG)	
Resource	Desktop * Anonymous volunteer	Desktop * within a corporation, university, institute, etc.	Supercomputer, cluster, scientific instruments, database, storage * Virtual organization (VO)
Connection	-Non dedicated poor bandwidth -Immediate presence (connectivity) -Consider firewall, NAT, Dynamic address	-Non dedicated intermediate bandwidth -More constant connectivity than volunteer DG	Dedicated high speed, bandwidth
Heterogeneity	High heterogeneous	Intermediate heterogeneous * Less heterogeneous than volunteer DG	Low heterogeneous
Dedication	-Non-dedicated -High volatile * Need an incentive mechanism	-Non-dedicated -Low volatile (non-business hours) * Need an incentive mechanism	Dedicated * Be able to use reservation
Trust	Malicious volunteer * Need result certification	Low trustworthy resource provider	High trustworthy resource provider
Failure	Unreliable (faulty)	Unreliable * More reliable than volunteer DG	More reliable than desktop grid
Manageability	Individual-based administration * Totally distributed to individual * Difficult to manage	Individual-based administration * More controllable than volunteer DG	-Domain-based administration * Professional administrator
Application (job)	-Independent (mainly) -Computation-intensive (mainly) -High-throughput (mainly)	-Independent (mainly) -Computation-intensive (mainly) * Data-intensive (possible) -High throughput (mainly)	-Independent or dependent -Computation or data-intensive -High performance (mainly)

Grid resources. Desktop Grid resources are administrated by individual users, whereas Grid resources are managed by professional administrator. The applications of Desktop Grid have no dependency between tasks. However, Grid includes dependent applications as well. Desktop Grid tries to achieve high throughput (that is, the amount of work that desktop computers can do within a given time period), whereas Grid mainly focuses on high performance (that is, the speed that a set of tasks run).

2.3 Taxonomy of Desktop Grid

Desktop Grid is categorized according to organization, platform, scale, and resource properties (see Figure 2.2).

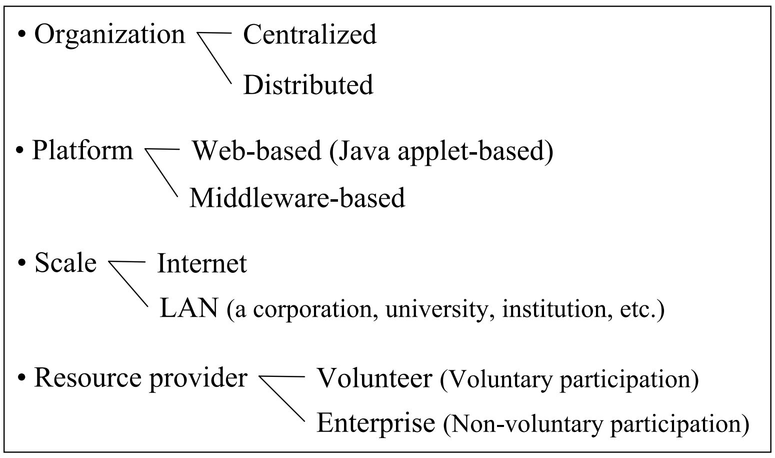

• Organization
Centralized
Distributed
• Platform
Web-based (Java applet-based)
Middleware-based
• Scale
Internet
LAN (a corporation, university, institution, etc.)
• Resource provider
Volunteer (Voluntary participation)
Enterprise (Non-voluntary participation)

Figure 2.2: A taxonomy of Desktop Grid

2.3.1 Organization

Desktop Grid is categorized into centralized and distributed ones according to organization of components.

Centralized Desktop Grid

Centralized Desktop Grid (DG) consists of client, volunteer, and server. The execution model of centralized DG consists of seven phases: registration, job submission, task allocation, task execution, task result return, result certification, and job result return phase as shown in the Figure 2.3.

- ***Registration phase***: Volunteers register their information to a server.
- ***Job submission phase***: A client submits a job to a server.
- ***Task allocation phase***: A server distributes tasks to the registered volunteers by means of a scheduling mechanism.
- ***Task execution phase***: Each volunteer executes its task.
- ***Task result return phase***: Each volunteer returns the result of its task to the server.
- ***Result certification phase***: The server checks the correctness of the returned results in order to tolerate malicious volunteers

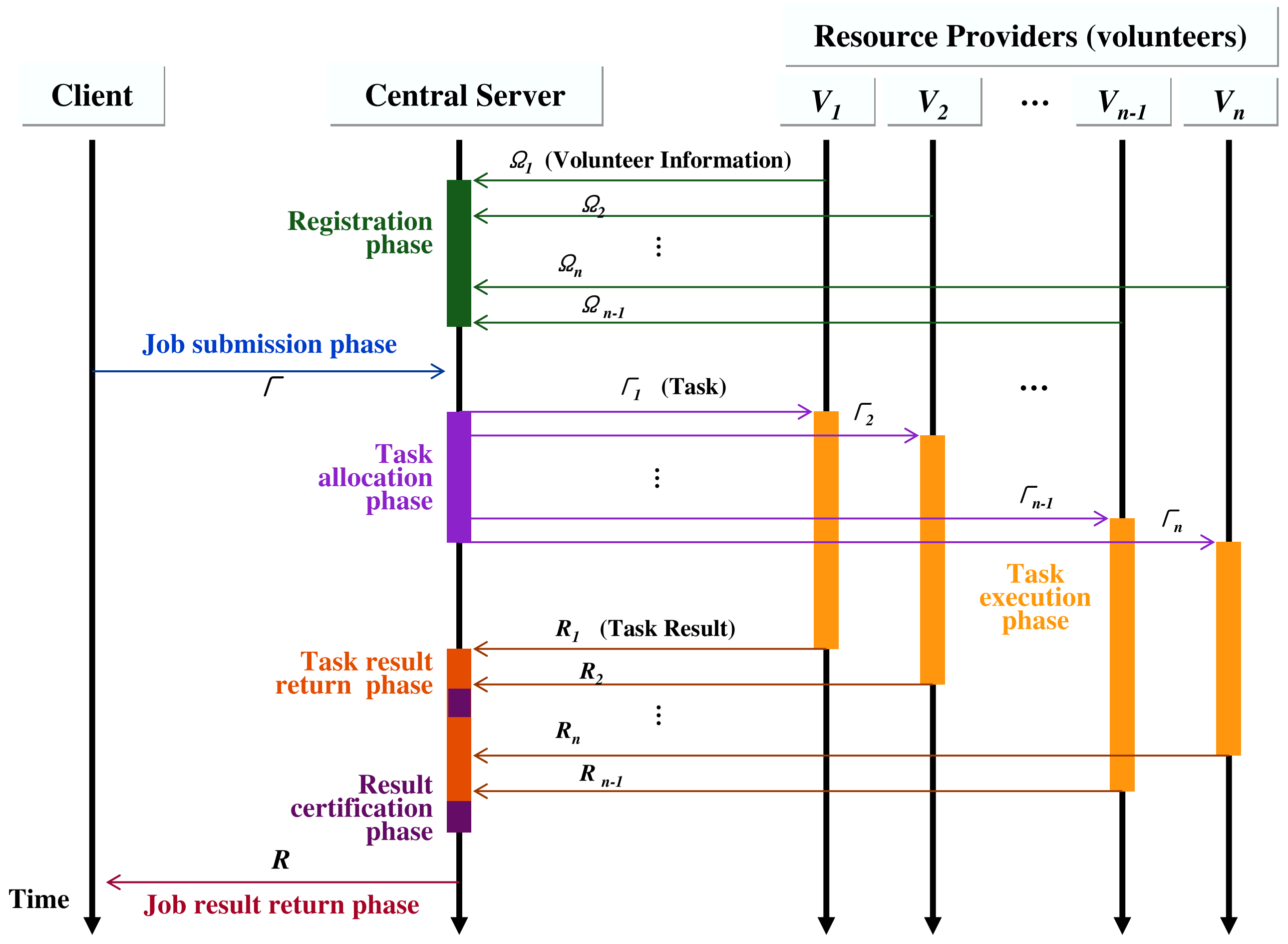


Figure 2.3: Execution model of centralized Desktop Grid

[23, 56, 98], or to deal with variations in numerical processing due to a variety of hardware and software [15].

- ***Job result return phase***: The server returns the final result of the job to the client.

Typical examples are BOINC [13, 14, 16], XtremWeb [17, 18, 19], Entropia [20, 21], Bayanihan [22, 23, 24], Korea@Home [52], and so on.

Distributed Desktop Grid

Distributed Desktop Grid[1] (DG) consists of client and volunteer. In contrast to centralized DG, there is no server, so volunteers have the partial information of other volunteers. Volunteers are responsible for constructing computational overlay network (CON) [6, 7, 8, 9]. The CON[2] is a set of volunteers for the execution of tasks. Scheduling is performed at each volunteer in a distributed way, depending on CON. That is, volunteers distributes tasks to other volunteers differently according to a characteristic or topology of CON (for example, tree, graph, or DHT(Distributed Hash Table)).

[1]Distributed Desktop Grid can be also called Peer-to-Peer (P2P) Desktop Grid in the sense that it constructs computational overlay network and performs scheduling by using Peer-to-Peer communication [6, 7, 8, 9]. Centralized Desktop Grid also can be called a P2P Grid if it uses peer to peer technologies to perform scheduling, resource management, or resource grouping [54, 55, 56, 57].

[2]CON construction is similar to resource grouping except that CON construction is mainly performed at a broker or a volunteer in a distributed manner. Resource grouping is mainly performed at a server in a centralized way.

The execution model of distributed DG consists of seven phases: registration, job submission, CON construction, task allocation, task execution, task result return, and job result return phase as shown in the Figure 2.4.

Typical examples are Javelin [25, 26, 27], Computer Power Market (CPM) [28, 29], CCOF [34, 35], Organic Grid [38, 39], Messor [40, 41], Paradropper [42, 43].

- ***Registration phase***: Volunteers exchange their information between other volunteers.[3]
- ***Job submission phase***: A client consigns a job to its neighbor volunteers.
- ***CON construction phase***[4]: Volunteers self-organize their CON according to capability, registration time, timezone, or randomly in a distributed way.[5]
- ***Task allocation phase***: Volunteers distribute tasks to the neighbors or appropriate volunteers by using a distributed scheduling mechanism.

[3]Volunteers can register their information to brokers [25, 28].

[4]A CON can be constructed on-the-fly or before-scheduling. In the case of on-the-fly, CON construction and scheduling are performed at the same time [38, 41, 42]. In the case of before-scheduling, CON construction is performed before scheduling [25]. Scheduling is performed on the basis of the structure of CON.

[5]A broker can be responsible for construction of CON (that is, tree) [25].

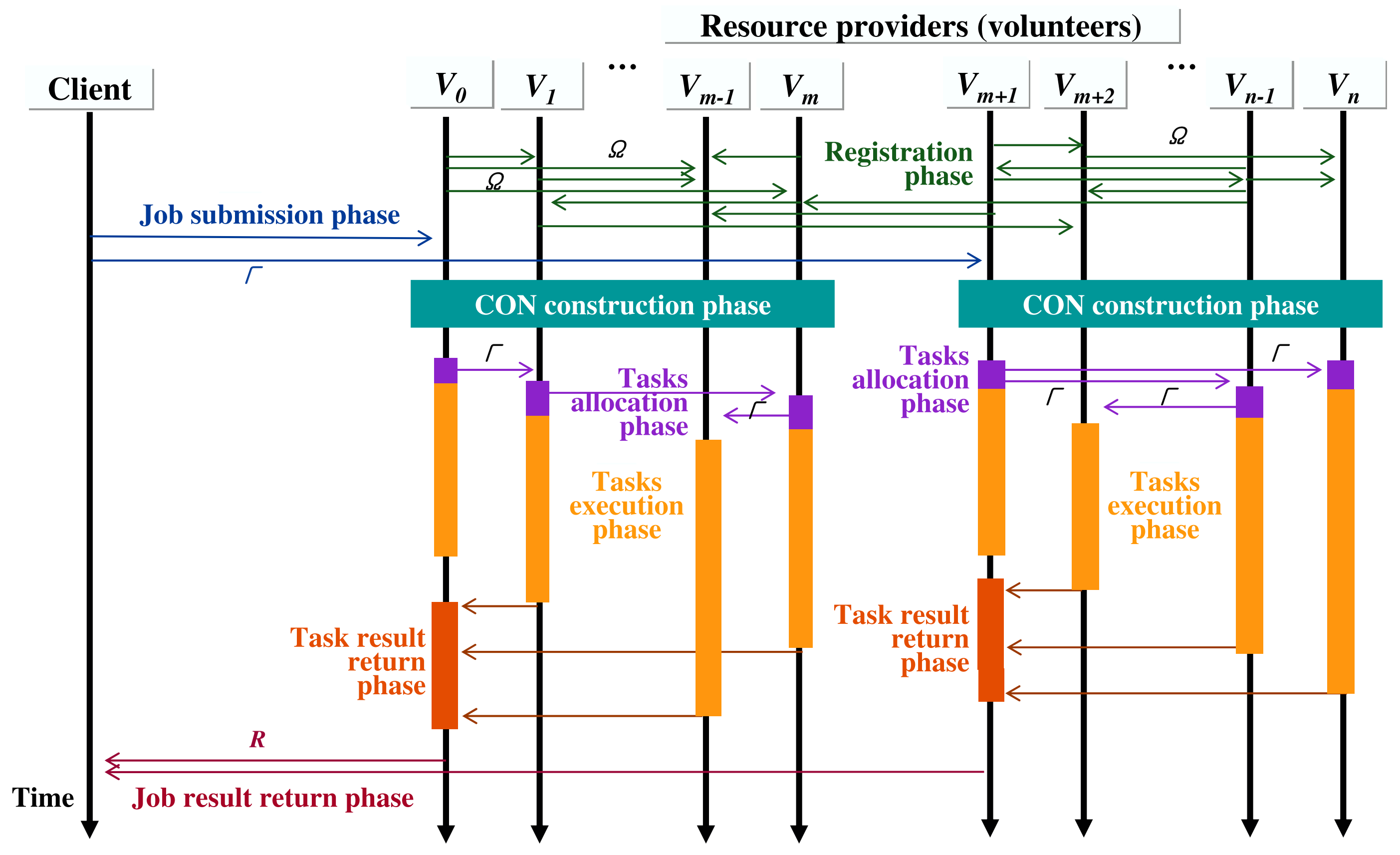


Figure 2.4: Execution model of distributed Desktop Grid

- ***Task execution phase***: Each volunteer executes its task.
- ***Task result return phase***: Each volunteer returns the result of its task to its parent volunteer.
- ***Job result return phase***: The parent volunteers return the final results of the jobs to the client.

2.3.2 Platform

Desktop Grid is categorized into web-based (Java applet-based) DG and middleware-based DG according to platform running on volunteers.

In the web-based DG, clients write their parallel applications in Java and post them as Applet on the Web. Then, volunteers only join the web page with their browsers. The Applet is downloaded automatically and runs on the volunteer's machine. Typical examples are Charlotte [30], Bayanihan [22, 23, 24], Javelin [25, 26, 27], and so on.

In the middleware-based DG, volunteers need to install and run a specific middleware (software that provides the services and functionalities to execute parallel applications) on their machine. The middleware automatically fetches tasks from a server and executes them, when CPU is idle. Typical examples are BOINC [13, 14, 16], XtremWeb [17, 18, 19], Entropia [20, 21], Bayanihan [22, 23, 24], Korea@Home [52], and so on.

2.3.3 Scale

Desktop Grid is categorized into Internet-based DG and LAN-based DG according to scale. Internet-based DG is based on anonymous volunteers (see Table 2.1). It should consider firewall, NAT, dynamic address, poor bandwidth, and unreliable connection. On the other hand, LAN-based DG is based on volunteers within a corporation, university, and institution. It has more constant connectivity than Internet-based DG.

2.3.4 Resource Provider

Desktop Grid is categorized into volunteer DG and enterprise DG according to properties of resource provider (see Table 2.1). Volunteer DG is mainly based on voluntary participants. Enterprise DG is mainly based on nonvoluntary participants usually within a corporation and university. Mostly, volunteer DG can be Internet-based DG, and enterprise DG can be LAN-based DG. Volunteer DG is more volatile, malicious, and faulty than enterprise DG. Enterprise DG is more controllable than volunteer DG because volunteers are located in the same administrative domain. Typical examples of volunteer DG are BOINC [7, 8, 9], XtremWeb [10, 11], Bayanihan [14, 15, 16], Javelin [17, 18], Korea@Home [38], and so on. Enterprise DG can be Entropia [13] and Condor [32, 33].

2.4 Taxonomy of Desktop Grid Scheduling

The taxonomy of scheduling for Desktop Grid is defined by three aspects: application, resource, and scheduler's perspective.

2.4.1 Application's Perspective Considerations

A Desktop Grid scheduler should consider the following aspects on the application's perspective when designing and developing scheduling algorithm (See Figure 2.5).

- ***Dependency***: Is there dependency between tasks? In the case of dependent tasks, the relationship between tasks are mainly designed as a graph (for example, Directed Acyclic Graph (DAG)) [70, 86]. The scheduler for DAG considers machine's capability, communication cost, data and task dependency (synchronization requirement) simultaneously in order to minimize the overall execution time of the graph [70, 85]. In the case of independent tasks, a scheduler focuses on allocation of independent tasks to machines according to resource's availability, capability, and properties [14, 25, 36, 49, 54, 67, 68].

- ***Type***: Is an application computation-intensive or data-intensive? In the case of data-intensive, a scheduler should consider location of data or replica, the cost of transfer, or replication policy, and so

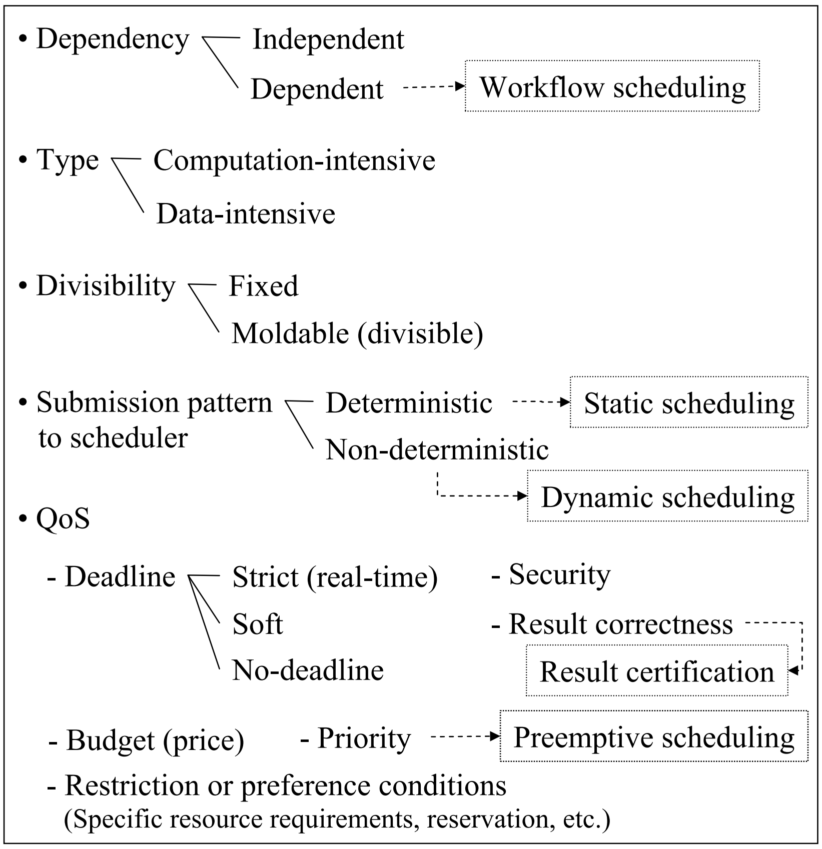


Figure 2.5: Application's perspective considerations

on [59]. In the case of computation-intensive, a scheduler focuses on resource's capability and availability.

- ***Divisibility***: Is a job divided into multiple subjob (task) flexibly? In case of divisible or moldable job, a scheduler focuses on how much a task are assigned to a resource [64]. The size of task depends on resource's capability, availability, deadline, etc.

- ***Submission pattern to scheduler***: Does a client submit an application to scheduler before scheduling? Or does a client non-deterministically submit an application to scheduler during scheduling?

- ***QoS***: Some applications may request QoS to a scheduler. For example, a certain application needs to be finished before the deadline. Assume that an application with highest priority wants to process more immediately and quickly than other applications. In this case, a high-priority application can preempt a low-priority application. A certain application does not want to be assigned to specific nodes or domain. A certain application needs to guarantee result correctness.

2.4.2 Resource's Perspective Considerations

A Desktop Grid scheduler should consider the following aspects on the resource's perspective when designing and developing scheduling algorithm (See Figure 2.6).

- ***Dedication to public execution (or volatility)***: Are resources allowed to freely join and leave in the middle of the public executions (that is, the execution of a task as a volunteer) without any constraints? If resources are volatile and non-dedicated, public execution can be suspended or stopped by a private execution (that is, the execution of a private job as a personal user). In

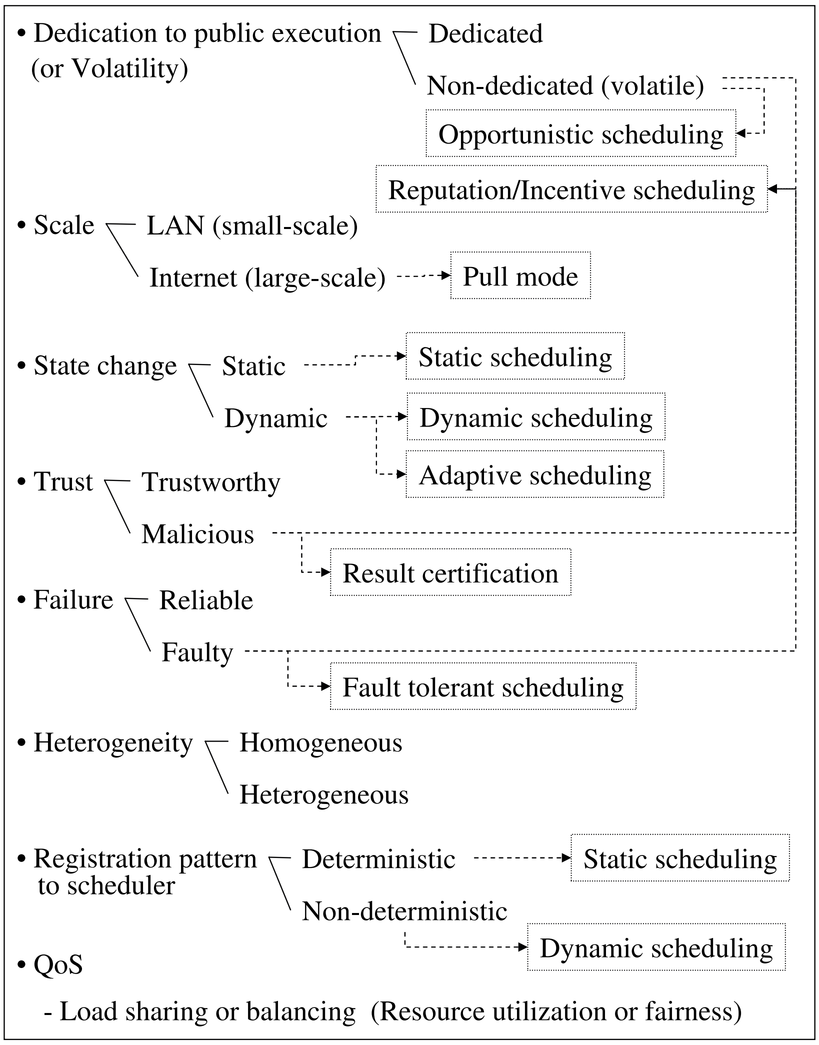


Figure 2.6: Resource's perspective considerations

this case, it is appreciate that a scheduler is opportunistic in the sense that a resource is not always available. A scheduler can be coupled with reputation or incentive mechanism in order to select eager resources or exclude selfish resources [54, 80, 81].

- ***Scale***: Are resources located in the scope of LAN or Internet? As shown in Table 2.1, the characteristics of environment (such as connection, the degree of heterogeneity and trust, dedication pattern, failure, manageability, etc.) are different between Internet-based DG and LAN-based DG. If resources are connected to Internet, it is proper that scheduling event is initiated by a resource's request in the sense that some resources are behind network-address translator (NAT) or firewall and they are not always available [13, 14, 52]. In other words, a resources pulls a task from its scheduler (that is, pull mode).

- ***State change***: Is the properties of resources (such as availability, volatility, trust, failure, load, bandwidth, etc.) changing during the public execution? In a Desktop Grid environment, resources are controlled by individual owners. Resources are more heterogeneous, dynamic and unreliable, when compared to Grid. A Desktop Grid scheduler should be dynamic and adaptive. In other words, a scheduler should be able to change the scheduling policy, adapting to such a changing environment.

- ***Trust***: Are resources trustworthy or malicious? If they are malicious, a scheduler needs result certification in order to ensure the correctness of results. A scheduler can be coupled with reputation or incentive mechanism in order to select trustworthy resources or exclude malicious resources [56, 80, 81, 82].

- ***Failure***: Are resources reliable or faulty? If they are faulty, a scheduler needs fault tolerant scheduling (that is, checkpoint & restart, reassignment, replication, etc.). Particularly, a scheduler in Desktop Grid should take volatility into account because volatility leads to the delay and blocking of the executions of tasks and even partial or entire loss of the executions. A scheduler can be coupled with reputation or incentive mechanism in order to select reliable resources or exclude faulty resources.

- ***Heterogeneity***: Resource heterogeneity refers to capability heterogeneity (that is, CPU, memory, bandwidth, OS type, etc.) as well as execution heterogeneity (that is, availability, credibility, volunteering time, the number of the completed tasks, etc.). Particularly, the execution heterogeneity makes scheduling more difficult in Desktop Grid. It is proper that a scheduler is coupled with resource grouping by which resources with similar properties are grouped together, in the sense that a scheduler can apply scheduling, fault tolerance, and result certification algorithms suitable for each group [54, 55, 56, 57].

- ***Registration pattern to scheduler***: Do resources participate in public executions deterministically or arbitrarily? If the information about resources is assumed to be unavailable before scheduling decision, or if resources freely join or leave the public execution, a dynamic scheduling approach is used [54, 61, 62, 63, 67].

- ***QoS***: Load sharing and balancing are necessary for resource utilization and fairness as well as performance. Load sharing aims to avoid having idle resources as much as possible by distributing the workload, whereas load balancing attempts to equalize workload among resources [76, 77, 78]. Work stealing and load redistribution (that is, transferring tasks from heavily-loaded node to lightly-loaded node) improve resource utilization, fairness, and performance.

2.4.3 Scheduler's Perspective Considerations

A Desktop Grid scheduler should consider the following aspects on the scheduler's perspective when designing and developing scheduling algorithm (See Figure 2.7 and 2.8).

- ***Organization***: A scheduler organization is classified into three categories: centralized, distributed, and hierarchical according to where and how scheduling decision is made [69, 72]. In the **centralized approach**, there is a central server that is responsible for scheduling decision. A central server maintains all information of resources and task execution status. In the **distributed approach**, scheduling decision is distributed to every node. Each node has the partial information about the resources and task execution status. In the **hierarchical approach**, the scheduling de-

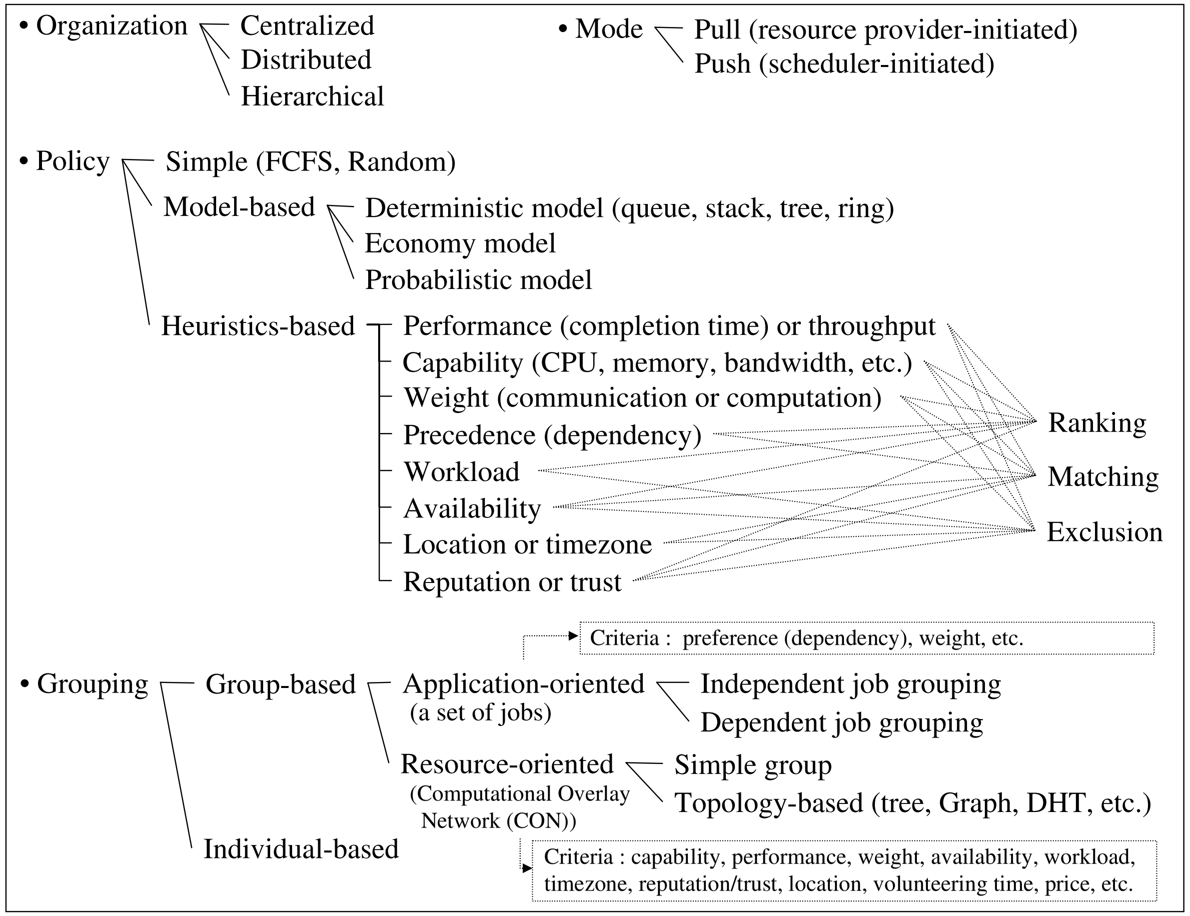

• Organization
Centralized
Distributed
Hierarchical
• Mode
Pull (resource provider-initiated)
Push (scheduler-initiated)
• Policy
Simple (FCFS, Random)
Model-based
Deterministic model (queue, stack, tree, ring)
Economy model
Probabilistic model
Heuristics-based
Performance (completion time) or throughput
Capability (CPU, memory, bandwidth, etc.)
Weight (communication or computation)
Precedence (dependency)
Workload
Availability
Location or timezone
Reputation or trust
Ranking
Matching
Exclusion
Criteria : preference (dependency), weight, etc.
• Grouping
Group-based
Application-oriented
(a set of jobs)
Independent job grouping
Dependent job grouping
Resource-oriented
(Computational Overlay Network (CON))
Simple group
Topology-based (tree, Graph, DHT, etc.)
Individual-based
Criteria : capability, performance, weight, availability, workload, timezone, reputation/trust, location, volunteering time, price, etc.

Figure 2.7: Scheduler's perspective considerations

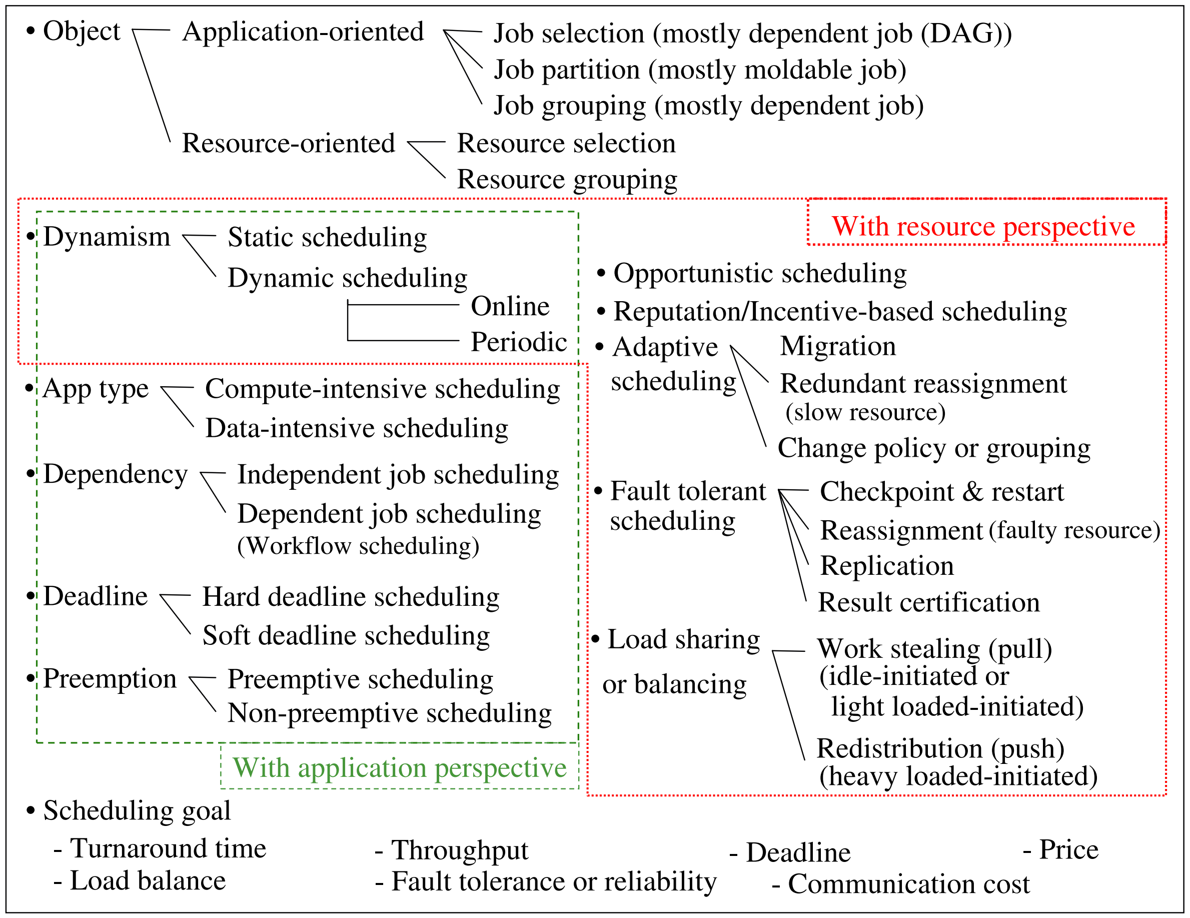

• Object
Application-oriented
Job selection (mostly dependent job (DAG))
Job partition (mostly moldable job)
Job grouping (mostly dependent job)
Resource-oriented
Resource selection
Resource grouping
With resource perspective
• Dynamism
Static scheduling
Dynamic scheduling
Online
Periodic
• App type
Compute-intensive scheduling
Data-intensive scheduling
• Dependency
Independent job scheduling
Dependent job scheduling (Workflow scheduling)
• Deadline
Hard deadline scheduling
Soft deadline scheduling
• Preemption
Preemptive scheduling
Non-preemptive scheduling
With application perspective
• Opportunistic scheduling
• Reputation/Incentive-based scheduling
• Adaptive scheduling
Migration
Redundant reassignment (slow resource)
Change policy or grouping
• Fault tolerant scheduling
Checkpoint & restart
Reassignment (faulty resource)
Replication
Result certification
• Load sharing or balancing
Work stealing (pull) (idle-initiated or light loaded-initiated)
Redistribution (push) (heavy loaded-initiated)
• Scheduling goal
- Turnaround time
- Load balance
- Throughput
- Fault tolerance or reliability
- Deadline
- Communication cost
- Price

Figure 2.8: Scheduler's perspective considerations (continued)

cision is performed in a hierarchical way (for example, meta scheduler (high-level scheduler) and local scheduler (low-level scheduler)). High-lever scheduler allocates tasks to low-level schedulers, whereas low-lever scheduler directly allocates tasks to machines within its site.

- ***Mode***: Where is a scheduling event initiated? In the **pull mode**, a scheduling event is initiated resource [19, 21, 74]. In other words, when a resource is idle or highly-loaded, it requests (or pulls) tasks from its server. In the **push mode**, a scheduler collects resource information, and then pushes tasks to resources [74]. Generally, the pull mode is useful if resources may be behind NAT(network-address translators) or firewall, or if they are not dedicated or volatile [14, 21, 54, 74].

- ***Policy***: Scheduling policy is used to match tasks with resources [61, 62, 63, 64, 67, 68, 69, 70, 71, 72, 79, 84]. It determines how to select appropriate tasks or resources. It is classified into three categories: simple, model-based, and heuristics-based. In the **simple approach**, tasks or resources are selected by using FCFS (First Come First Served) or randomly. The **model-based approach** is categorized into deterministic, economy, and probabilistic models. The deterministic model is based on structure or topology such as queue, stack, tree, or ring. Tasks or resources are deterministically selected according to the properties of structure or topology.

For example, in a tree topology, tasks are allocated from parent nodes to child nodes. In the economy model, scheduling decision is based on market economy (that is, price and budget). In the probabilistic model, resources are selected in probabilistic manners (such as Macov, machine learning, or genetic algorithms). In the **heuristics-based approach**, tasks or resources are selected by ranking, matching, and exclusion methods on the basis of performance, capability, weight, precedence, workload, availability, location, reputation/turst, etc. The ranking method ranks the resources or tasks according to criteria and then chooses the most or the worst one. The matching method chooses the most suitable tasks and resources in accordance to evaluation functions (for example, min-min, max-min, sufferage, etc. [67, 68, 84]). The exclusion method excludes resources according to criteria, and then chooses the most appropriate one among the survivors. Ranking, matching, and exclusion methods can be used together or separately. In the case of criteria, precedence is used only for tasks. Workload, availability, location/timezone, and reputation/turst are used for resources. Performance, capability, and weight are used for both resources and tasks.

- ***Grouping***: Grouping is used to form resources or tasks into a group. In the **application-oriented grouping approach**, a set of jobs are grouped logically [70, 86]. Particularly, dependent tasks are grouped together on the basis of dependency or weight (communication or computation) in DAG in order to reduce communication cost or improve performance [70, 86]. In addition, tasks are grouped together so that a set of tasks that uses the same data is allocated to one node [59, 86]. The **resource-oriented grouping approach** ensures that resources with similar properties are logically grouped together. Resource-oriented grouping approach constructs CON (Computational Overlay Network). The characteristics and topology of CON affect scheduling algorithm, resource management, and information management. As a result, the reliability, result correctness, and performance depend on how the CON is constructed. Performance and reliability can improve by applying suitable scheduling, fault tolerance, and result certification algorithms to each group. A CON is categorized into simple group and topology-based. In the simple group approach, resources are grouped together according to capability, performance, weight, availability, workload, reputation/trust, volatility, and so on. In the topology-based approach, resources are grouped together while forming topologies such as tree, graph, or DHT (Distributed Hash Table).

- ***Object***: Scheduling decision is made in an application-oriented or resource-oriented way according to the target of scheduling. The **application-oriented approach** focuses on job selection, partition, and grouping. On the other hand, **resource-oriented approach** emphasizes resource selection and grouping. A dependent job (or DAG) is mainly related with application-oriented approach (that is, which task is first processed, or how tasks are grouped or divided for a resource) [70, 86], whereas an independent job is mainly related with resource-oriented approach (that is, to decide which resource is appropriate for a task) [67, 68, 86].

- ***Dynamism***: Scheduling is categorized into static and dynamic according to whether the information of jobs and resources is known or available, and when scheduling decision is made [61, 62, 65, 66, 67]. In the case of **static scheduling**, the prior information is assumed to be available [61, 62, 65, 66, 67]. Static scheduling considers the entire tasks during decision making. In the case of **dynamic scheduling**, little a prior knowledge is available [61, 62, 65, 66, 67]. It is unknown in what environment tasks will execute. In addition, some nodes may go off-line and new nodes may come on-line. That is, the environment state is changing over time. Dynamic scheduling obtains dynamically changing state and then takes the environment inputs into account when making decisions. Dynamic scheduling can involve adaptive scheduling, fault-tolerant scheduling, and load sharing and balancing. Dynamic scheduling is classified into online and periodic according to the time at which scheduling event occurs [67].

In the online approach, scheduling event occurs as soon as a task or a resource arrives. In the periodic approach, scheduling event occurs periodically (that is, every predefined interval or time).

- ***Application type***: Compute-intensive scheduling focuses on how tasks are assigned to resources according to resources' properties. Data-intensive scheduling focuses on data such as data size and location, the cost of transfer, replication policy, or data dependency.

- ***Dependency***: Dependent job scheduling (that is, workflow scheduling) focuses on task and data dependency and synchronization between tasks in order to minimize the overall execution time of DAG [70, 85]. Independent job scheduling focuses on distribution of independent tasks to each machine according to machine's availability and capability, in order to complete as many tasks as possible, concurrently.

- ***Deadline***: Deadline scheduling distributes tasks to resources only if the resources are able to (that is, hard deadline) or are likely to (that is, soft deadline) complete the task by its deadline.

- ***Preemption***: Preemptive scheduling considers task's priority. It allows a high-priority task to preempt a low-priority task running on a machine. In non-preemptive scheduling, a machine is allowed to execute another task only after finishing a task.

- ***Opportunistic scheduling***: Opportunistic scheduling is to use resources as quickly as possible, when they are idle or available [30, 45, 54]. It can be easily cooperated with the pull mode.

- ***Reputation/incentive-based scheduling***: Reputation/incentive-based scheduling evaluates resources in order to select more-qualified resources [56, 80, 81, 82]. If resources are selfish (non-dedicated), distrusted, volatile, or faulty, a reputation-based or an incentive-based scheduling is needed to exclude these resources or to encourage resource's owners to provide their resources reliably, eagerly, and trustworthily.

- ***Adaptive scheduling***: Adaptive scheduling takes environmental stimuli into account to adapt to dynamically changing environment [61, 62, 78]. The environmental change leads to modifying the scheduling policy. Adaptive scheduling is classified into migration, redundant reassignment, and change-policy or topology. In the **migration approach**, a task is moved from one node to another node. In the **redundant reassignment approach**, the task that a slow resource does not complete within timeout is reassigned to other resources. This leads to replication. In the **change-policy or topology approach**, scheduling policy or topology is switched in accordance with environmental change. For example, in a tree topology, fast nodes move towards a root node. Or, in the SA (Switching Algorithm), MCT (Minimum

Completion time) heuristic is switched to MET (Minimum Execution Time) depending on the load distribution threshold across the nodes [67].

- ***Fault tolerant scheduling***: Fault tolerant scheduling tolerates failure as well as volatility. It is classified into checkpoint & restart, reassignment, replication, and result certification. In the **checkpoint & restart approach**, if a scheduler detects failure of resource, it restarts the failed task at another resources from the checkpoint. In the **reassignment approach**, if a scheduler detects failure of resources, it reassigns the failed task to another node. In the **replication approach**, a scheduler replicates the same task to multiple nodes. Even though one of them fails, the others mask the failure. **Result certification approach** tolerates malicious resources or a variety of hardware and software malfunctions [15, 23, 56, 98]. As a result, it guarantees the correctness of results.

- ***Load sharing or balancing***: Load sharing or balancing is categorized into work stealing and redistribution. In the **work stealing approach**, a lightly-loaded node or idle node steals (or pulls) tasks from a heavily-loaded node. On the contrary, in the **redistribution approach**, a heavily-loaded node transfers (or pushes) tasks to a lightly-loaded node or idle node.

- ***Scheduling goals***: A scheduler tries to achieve its own scheduling goals. It chooses appropriate scheduling policies and algorithms according to its goals such as turnaround time, throughput, deadline, price, load balance, and reliability.

2.5 Survey of Desktop Grid Systems

A mapping of the taxonomy to existing Desktop Grid systems or projects is illustrated in this section. Table 2.2 shows the survey of existing Desktop Grid systems according to the taxonomy of Desktop Grid shown in Figure 2.2. Tables 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 show the survey of existing Desktop Grid systems focusing on scheduling according to the taxonomy of scheduling shown in Figures 2.7 and 2.8.

We survey existing Desktop Grid systems, projects, and papers: BOINC [13, 14, 15, 16], XtremWeb [17, 18, 19], Entropia [20, 21], Bayanihan [22, 23, 24], Javelin [25, 26, 27], Computer Power Market (CPM) [28, 29], Charlotte [30], POPCORN [31], WebCom [32, 33], Cluster Computing On the Fly(CCOF) [34, 35, 36, 37], Organic Grid [38, 39], Messor [40, 41], Paradropper [42, 43], and Condor [44, 45, 46].

Table 2.2: A Survey of Desktop Grid

System	Organization	Platform	Scale	Resource provider
BOINC	Centralized	Middleware-based	Internet	Volunteer Or Enterprise
XtremWeb	Centralized	Middleware-based	Internet	Volunteer
Entropia	Centralized	Middleware-based	LAN or Internet	Enterprise or Volunteer
Bayanihan	Centralized	Web-based or Middleware-based	Internet	Volunteer
Javelin	Distributed	Web-based or Middleware-based	Internet	Volunteer
CPM	Distributed	Middleware-based	Internet	Volunteer
Charlotte	Centralized	Web-based	Internet	Volunteer
Popcorn	Centralized	Web-based	Internet	Volunteer
WebCom	Centralized	Web-based	Internet	Volunteer
CCOF	Distributed	Middleware-based	Internet	Volunteer
Organic Grid	Distributed	Middleware-based	Internet	Volunteer
Messor (Anthill)	Distributed	Middleware-based	Internet	Volunteer
Paradropper	Distributed	Middleware-based	Internet	Volunteer
Condor	Centralized	Middleware-based	LAN	Enterprise

2.5.1 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [13, 14, 15, 16] is a well-known middleware system for volunteer computing (or public-resource computing). BOINC makes it easy for scientists to create and operate public-resource computing projects. There are a lot of BOINC-based projects: SETI@Home, Predictor@Home, Folding@Home, Climatepredication.net, Climate@Home, LHC@Home, Einstein@Home, BBC Climate Change, and so on [13, 14, 15, 16].

Table 2.3: Survey of Desktop Grid systems focusing on scheduling

System	Organization	Mode	Policy	Grouping	Object	Dynamism	Etc.
BOINC	Centralized	Pull	FCFS	Individual-based (Not supported)	Resource oriented: resource selection	Dynamic	-SETI@Home, Einstein@Home, Climatepredict.net
XtremWeb	Centralized	Pull	FCFS	Individual-based	Resource oriented : selection	Dynamic	Java based DG
Entropia	Centralized	Pull	-Model-based: Queue (job selection) -Capability :matching (resource selection, grouping)	Resource-oriented : Simple grouping (capability)	Resource oriented: resource selection, grouping	Dynamic	-Volunteer DG (ENTROPIA 2000) -Enterprise DG (DCGRID^{TM})
Bayanihan	Centralized	Pull	-FCFS -Reputation/trust: exclusion (Credibility)	Individual-based	Resource oriented: resource selection	Dynamic	-Credibility based eager scheduling -Java based volunteer computing
Javelin	Distributed	Pull	-Random -Deterministic model: tree (volunteer)	Resource-oriented : CON (tree)	Resource oriented: resource selection, grouping	Dynamic	-Tree based advanced eager scheduling -Java-based
CPM	Distributed	Pull	Economy model	Individual-based	Resource oriented: resource selection	Dynamic	Market-oriented
Charlotte	Centralized	Pull	FCFS	Individual-based	Resource oriented: resource selection	Dynamic	Eager scheduling
Popcorn	Centralized	Pull	Economy-model	Individual-based	Resource oriented: resource selection	Dynamic	Online market based scheduling

Table 2.4: Survey of Desktop Grid systems focusing on scheduling (Continued)

System	App. Type	Dependency	Deadline	Preemption
BOINC	Compute-intensive	Independent	Soft deadline	Preempt (if a volunteer participates in multiple projects)
XtremWeb	Compute-Intensive	Independent	Not mentioned	Not support
Entropia	Compute-Intensive	Independent	Soft deadline	Not support
Bayanihan	Compute-intensive	-Independent -Dependent (BSP)	Not mentioned	Not support
Javelin	Compute-intensive	Independent	Not mentioned	Not support
CPM	Compute-intensive	Independent	Soft deadline	Not support
Charlotte	Compute-intensive	Independent	Not mentioned	Not support
Popcorn	Compute-intensive	Independent	Not mentioned	Not support

Table 2.5: Survey of Desktop Grid systems focusing on scheduling (Continued)

System	Opportunism	Reputation / Incentive	Adaptiveness	Fault tolerance	Load balancing	Goal
BOINC	Opportunistic scheduling	Not support	Reassignment	-Checkpoint/restart -Result certification (homogeneous redundancy) -Redundant computing	No	-Fault tolerance -Reliability
XtremWeb	Opportunistic scheduling	Not support	Reassignment	Reassignment	No	Not specified
Entropia	Opportunistic scheduling	Not support	Reassignment	Reassignment	No	Not specified
Bayanihan	Opportunistic scheduling	Credibility-based eager scheduling	Reassignment	-Reassignment -Result certification (majority voting, spot-checking)	No	Fault tolerance (sabotage tolerance)
Javelin	Opportunistic scheduling	Not support	Reassignment	Reassignment	Work stealing (Pull)	-Fault tolerance -Load balance
CPM	Contract-based (price, deadline)	Not support	No	Reassignment	No	-Price -Deadline
Charlotte	Opportunistic scheduling	Not support	Reassignment	Reassignment	No	-Fault tolerance
Popcorn	Not mentioned	Not support	No	-Reassignment -Result Certification	No	Price

Table 2.6: Survey of Desktop Grid systems focusing on scheduling (Continued)

System	Organization	Mode	Policy	Grouping	Object	Dynamism	Etc.
WebCom	Hierarchical	Pull	-Deterministic : queue (round-robin) -Capability (network latency): matching -Performance: matching	Resource-oriented: CON (tree): latency	Resource oriented: resource selection	Dynamic	Condensed graph
CCOF	Distributed	Push	-Timezone: matching -Reputation/trust: ranking, exclusion	Resource-oriented: CON (Timezone based DHT)	Resource oriented: resource grouping, selection	Dynamic	Wave scheduling (night timezone based scheduling)
Organic Grid	Distributed	Pull	Deterministic model (tree)	Resource-Oriented: CON (Performance-based tree)	Resource oriented: resource grouping, selection	Dynamic	-Self-organizing scheduling -Mobile agent-based
Messor	Distributed	Push Pull	-Random -Workload: matching	Resource-oriented: CON (Workload-based Random graph)	Resource oriented: resource selection	Dynamic	-Ant (mobile agent) based load balancing
Paradropper	Distributed	Push	-Workload: ranking	Resource-oriented: CON (small world graph)	Resource oriented: resource grouping, selection	Dynamic	Small world graph
Condor	Centralized	Push, Pull	Capability, location, workload: matching, ranking	Individual-based	Resource oriented: selection	Dynamic	High throughput computing
Kondo et al.	Centralized	Pull, Push	-Capability (CPU): ranking, exclusion -Performance : exclusion -Throughput: ranking	Individual-based	Resource oriented: resource selection	Dynamic	-Enterprise DG -Availability measurement

Table 2.7: Survey of Desktop Grid systems focusing on scheduling (Continued)

System	App. Type	Dependency	Deadline	Preemption
WebCom	Compute-intensive	Dependent (Condensed graph)	Not mentioned	No
CCOF	Compute-Intensive	Independent	Soft deadline	No
Organic Grid	Compute-Intensive	Independent	Not mentioned	No
Messor	Compute-Intensive	Independent	Not mentioned	No
Paradropper	Compute-intensive	Independent	Not mentioned	No
Condor	Compute-intensive, Data-intensive	Independent, Dependent	Soft deadline	Preempt (in dedicated cluster)
Kondo et al.	Compute-intensive	Independent	Soft deadline	No

Table 2.8: Survey of Desktop Grid systems focusing on scheduling (Continued)

System	Opportunism	Reputation / Incentive	Adaptiveness	Fault tolerance	Load balancing	Goal
WebCom	Not mentioned	Not support	Change topology (tree) : load, latency	Reassignment	No	Fault-tolerance
CCOF	Opportunistic scheduling	Trust-based scheduling	Migration (timezone)	-Result certification (quiz and replication in collision)	No	-Turnaround time -Fault tolerance or reliability
Organic Grid	Opportunistic scheduling	Not support	Change topology (performance-based)	Reassignment	No	-Turnaround time
Messor	Opportunistic scheduling	Not support	Change policy *Push : overloaded *Pull : light loaded	No	-Work stealing (pull) -Load redistribution (push)	-Load balance
Paradropper	Opportunistic scheduling	Not support	No	No	Load redistribution (push)	-Load balance
Condor	-Matchmaking -Opportunistic scheduling	Not support	Migration	Checkpoint/restart	No	-Throughput -Fault tolerance
Kondo et al.	Not mentioned	Not support	Reassignment	Replication	No	-Turnaround time

BOINC consists of server and client (volunteer in this thesis). A server has a task server that dispatches tasks and processes the results of tasks, a data server that handles file transfer, a database that stores descriptions of applications, volunteers, scheduling, etc., and web interfaces for account management, message boards, etc. A client (that is, volunteer) runs projects' applications. It can participate in several projects and specify preferences for the projects. BOINC is mainly based on voluntary participants connected through Internet[6].

BOINC server is responsible for scheduling. A client sends a request to a task server (that is, pull mode). The server allocates a list of new tasks to the client. Additionally, BOINC supports locality scheduling. In addition, a client performs local scheduling on its computer[7], which decides which task to run among multiple projects' applications, when to ask a server for more works, which project to ask, and how much work to ask for [13]. BOINC clients can join and leave freely, so scheduling should be dynamic.

BOINC is used for applications in physics, molecular biology, medicine, chemistry, astronomy, climate, mathematics, etc. The applications are mainly compute-intensive and independent. BOINC scheduler distributes a task to a volunteer only if the volunteer is likely to complete the task

[6]Recently, BOINC can be used as Enterprise DG computing platform, although it was originally designed for volunteer DG (volunteer computing) [13].

[7]Local scheduling of BOINC is different from local scheduling of Grid. The local scheduler in Grid decides how to distribute or order tasks to multiple computers (that is, cluster) or processors in supercomputer

by its deadline. If an task's deadline passes or if a task fails, a server marks it as time-out and redistributes a new instance of the task. With the local scheduling, BOINC client may preempt applications either by suspending them or by instructing them to quit if it participates in multiple projects [13]. That is, one task is preempted by another task. BOINC provides checkpoint API for applications, that is, when to checkpoint and when a checkpoint has been done.

BOINC supports redundant computing to identify and reject erroneous results. It also provides homogeneous redundancy for the numerical applications that may produce different results depending on the machine architecture, operating systems, compiler, and compiler flags [15]. In this case, the redundant tasks are dispatched to numerically identical computers.

2.5.2 XtremWeb

XtremWeb [17, 18, 19] is a Java-based middleware system for global computing (large scale distributed system) experiments. XtremWeb extends the principle of cycle stealing to personal computers connected to Internet.

XtremWeb is composed of client, server (or coordinator), and worker (volunteer in this thesis). A client performs tasks submission and results retrieval. A coordinator is composed of a repository that advertises or publishes applications, a scheduler, a result server that collects results, and database. All the communications are initiated by a worker. A worker contacts its server to get tasks. That is, the scheduler uses pull mode to allocate tasks. The scheduler distributes tasks to workers in a FIFO (First In First Out) way.

The applications are mainly compute-intensive and independent. A worker periodically sends an alive signal to its server. If the server has not received the message during a predefined time (that is, timeout), it reschedules the task to another worker.

2.5.3 Entropia

Entropia [20, 21] is a middleware system for commercial Desktop Grid. Entropia provides two solutions: enterprise Desktop Grid (Entropia DC-Grid) and Internet Grid (Entropia 2000). The applications are mainly compute-intensive and independent.

Entropia consists of server and client (volunteer in this thesis). The server in Internet Desktop Grid consists of three main components: tasks server, file server, and app server. A task server is responsible for registration, scheduling and resource management. It maintains a database and forms an application pool (that is, a list of clients, the number of assigned jobs, and the number of completed jobs, and the pool priority). The clients within a pool have similar capabilities such as disk space or operating system type. The App server decomposes an job into subjobs and assigns them to clients. The task server performs scheduling in a pull mode.

Enterprise Desktop Grid has three layers: job management layer, subjob management layer, and resource management layer. The resource management layer is responsible for registration and resource management, and maintains resource pools (application pools). The subjob management layer performs scheduling. The job management layer's responsibility includes job decomposition and management. The subjob management layer maintains queues that have priority and default values for time to live, max time to run, and min time to run. It first processes the highest priority queue. Higher priority is assigned to the retried subjobs than first submitted subjobs. Subjobs are selected in a FIFO way. Like this, Entropia's subjob management layer provides how to select subjobs depending on queue structure. Clients periodically report their resource status to node manager in the resource management layer and the subjob scheduler. The scheduler assigns subjobs to available clients according to client's attributes such as memory capacity, OS type, etc. For example, if subjobs need a minimum of 128 MB of memory, then they are assigned to the clients with at least that amount of memory. If a client becomes disconnected or unresponsive, or fails to return a result within the expected time, the scheduler redistributes the subjob to another client.

2.5.4 Bayanihan

Bayanihan [22, 23, 24] is a web-based volunteer computing system using Java. Bayanihan system consists of client (volunteer in this thesis) and server. A client can either be Java applet started from a web browser (that is, web-based), or Java application (that is, middleware-based). A client has a worker engine for performing computation or a watcher engine for viewing results and statistics. A server consists of HTTP server, work manager, watch manager, and data pool. The HTTP server serves out Java class file. The work manager distributes tasks and collects result. The watch manager distributes results to watcher engines in clients.

The work manager in a server is responsible for scheduling. A worker client (that is, volunteer) makes remote call to the server to get a task. Bayanihan basically uses eager scheduling, in which a volunteer asks its server for a new task as soon as it finishes its current task. The more eager a volunteer works, the more tasks are executed. Additionally, it provides credibility-based fault tolerance mechanism to tolerate erroneous results from malicious volunteers [23]. The credibility-enhanced eager scheduling estimates the probability of result and worker being correct by using voting, spot-checking [23, 24]. In a majority voting approach, the same task is performed at different volunteers as much as the number of redundancy. Redundancy is used to identify the correct result against erroneous one if there are sufficiently more good volunteers than bad ones. In a spot-checking approach, the special task whose result is already known is performed at randomly selected volunteers. If a volunteer returns an erroneous result, it is regarded as malicious one. In the credibility-enhanced eager scheduling, the more a volunteer passes the spot-checking, the higher its credibility becomes. The more volunteers within voting group agree on a result, the higher its credibility becomes. Volunteers continue to compute the task and perform spot-checking until the credibility threshold is satisfied. When the desired credibility threshold is reached, the result is accepted as a final. The volunteers that produce erroneous results can be blacklisted. If a worker executes a task slowly or if it fails, the scheduler reassigns the task to different workers.

The applications are mainly compute-intensive and independent. In addition, Bayanihan supports applications running in BSP (Bulk Synchronous Parallel) mode [24], which provides familiar message-passing and remote memory primitives.

2.5.5 Javelin

Javelin [25, 26, 27] is a Java-based infrastructure for parallel Internet computing (or global computing). Applications run as Java applet (Javeline version) or screen saver (Javelin++ version). Applications are mainly compute-intensive and independent. Javelin consists of three entities: broker, client, and host (volunteer in this thesis). A client registers its tasks to a broker. A host offers computing resources. A broker coordinates the supply and demand for computing resources. When a host contacts a broker, the broker adds the host to a logical tree structure. A broker maintains the organized tree of hosts. Like this, Javelin simply constructs tree-based CON.

A client registers with a broker. If a host requests tasks to the broker, the broker informs the host of client ID and application information. Then the host executes tasks. At this time, work stealing and advanced eager scheduling are performed [26, 27]. With work stealing, when a host runs out of work, it request tasks from other hosts in two ways: deterministic or probabilistic approaches. In a deterministic approach, a host asks tasks from its children or its parents on the basis of tree structure. In a probabilistic approach, the host selects the target randomly from the list of hosts it currently knows. With advanced eager scheduling, the client selects the next task marked undone and reissues it to another host. The advanced eager scheduling is invoked only when work stealing fails. It also provides fault tolerant mechanism, that is, how to fix tree in the presence of host's failure. The failed work is redistributed by eager scheduling, in the sense that eager scheduling guarantees that the undone works will be rescheduled to different hosts eventually.

In Javelin, the work stealing is performed at a host, and eager scheduling is performed at a client. A broker is a simple mediator between clients and hosts. Thus, Javelin is close to distributed DG, although a broker collects hosts' information.

2.5.6 CPM

CPM(Compute Power Market) [28, 29] is a market-based middleware system for Grid computing on low-end personal computing devices connected to Internet. It aims to develop a computational marketplace for the regulation of resource demand and supply. It applies economic concept to resource management and scheduling of computations across Internet-wide volunteer resources.

CPM consists of a market, a resource consumer (client in this thesis), and a resource provider (volunteer in this thesis). A market is a mediator between consumer and provider. It maintains information about providers and consumers. A resource consumer buys computing power from the market. It has a market resource broker downloaded from the market. The market resource broker finds appropriate providers depending on the information provided by the market. It selects resources according to deadline or budget, negotiates the cost of resources, and distributes tasks to them. The application and data files are fetched when the task is ready to run at the resource provider. A resource provider sells computing power through the market. A resource provider has a market resource agent downloaded from the market. The market resource agent updates information about its resource provider, and deploys and executes tasks.

In CPM, resources trade is performed between consumer (that is, client) and producer (that is, volunteer). Scheduling is performed at a resource consumer (that is, client). Thus, CPM is close to distributed DG. Scheduling is also close to distributed one although a consumer is responsible for scheduling, in the sense that consumers negotiate for resource's cost with providers and there is no special server that is responsible for scheduling.

2.5.7 Charlotte

Charlotte [30] is a Java-based infrastructure for metacomputing on the Web. It consists of manager (server in this thesis) and worker (volunteer in this thesis). A manager provides a scheduling service and a memory service for accessing shared data. A worker provides a computing service implemented as an applet.

Charlotte firstly proposed eager scheduling. At first, participating workers pickup and execute each task. After then, if a worker finishes its task, it contacts the eager scheduler. If there are still tasks that have not been assigned, the scheduler assigns one of them to the worker. If all tasks have been assigned but some tasks have not yet completed, the scheduler reassigns one of the unfinished tasks to the worker. This redundant assignment of a task to multiple workers eventually tolerates slow workers and failed workers in the sense that if at least one of multiple workers finishes the task, the task is completed. That is, eager or fast workers overtake slow or failed workers.

2.5.8 POPCORN

POPCORN [31] is a Java-based infrastructure for globally distributed computation over the Internet. POPCORN provides a market-based mechanism for trade of computational resources. POPCORN system consists of market, seller (volunteer in this thesis), and buyer (client in this thesis). A market matches buyers and sellers according to economic model. A seller provides its resource to a buyer by using Java-enabled browser. The applications are mainly compute-intensive and independent.

A market uses the popcoin (that is, an abstract currency) and maintains a database about it. A seller can earn popcorn or get any other type of reward such as on-line game or a picture by barter. A buyer can buy resources with popcoins. The market is responsible for matching buyers and sellers, for transferring tasks and results between them, and for handling all payments and accounts. It is a well-known meeting place or matchmaker for buyers and sellers. It uses several market models (that is, a repeated Vickrey auction, a sealed bid double-auction, and repeated Clearinghouse double auction) for matching buyers and sellers [31].

POPCORN deals with the failure and verification. If a task fails, simply resend the tasks to a different host. POPCORN simply uses replication, spot-checking, self-testing, etc.[8]

2.5.9 WebCom

WebCom [32, 33] is a web-based distributed computation platform using Java. It consists of a master (server in this thesis) and a client (volunteer in this thesis). A master maintains a set of clients and is responsible

[8]It does not propose a new mechanism. It just uses existing mechanisms.

for scheduling. A client receives a task as the form of a Java applet, executes it within its browser.

A master generates atomic instruction (that is, a task) or a condensed graph (that is, a set of tasks) that represents an acyclic graph of interacting sequential programs. Execution begins at the master, and the tasks are distributed to clients when clients become available. A client can act as a potential master. It can be promoted to be a master if it receives a condensed graph. The promoted master is assigned a number of clients according to communication latency. If it needs more clients, it requests them to the primary master. Conversely, it redirects its clients to its primary master if they are under-utilized. Like this, the primary master, promoted masters, and clients form a tree structure. Scheduling is performed hierarchically at the primary master and the promoted masters.

A master maintains instruction queues for the atomic or condensed graph instructions. It distributes instructions to clients in a round robin fashion. It also performs scheduling according to network latency between client and master. It allocates tasks to clients on the basis of an expected execution time and CPU performance (load). If a task fails, it is rescheduled to another client.

2.5.10 CCOF

CCOF(Cluster Computing On the Fly) [34, 35, 36, 37] is a cycle sharing peer-to-peer system. It harvests idle cycles from users at the edges of the Internet. It supports a distributed model, in which there is no server and any peer can be either a donor or a consumer or both.

Hosts (volunteers in this thesis) join a community based overlay networks (CAN-based DHT overlay) to donate their idle cycles [34, 35]. Clients discover these resources from these overlays, and schedule tasks according to timezone. If a scheduler fails to find enough resources at day timezone, it reschedules the tasks at night timezone. If a host is not able to complete its task, the task migrates to a new host at night timezone for fast turnaround time[9]. If the host fails to find a new host, the original client reschedules the job. The scheduler is called wave scheduler in that tasks ride a wave of idle cycles (day or night timezone). Wave scheduling supports deadline-driven tasks. Scheduling is close to distributed model in the sense that there is no server and migration is performed at each host, although a client initially schedules the tasks.

CCOF provides result verification schemes (that is, Quiz and replication) on the assumption that there is collusion among malicious hosts [37]. The Quiz inserts a quiz task (its result is known to a client) into a normal task (that is, it is similar to spot-checking). CCOF also proposes a trust-based scheduling. The trust-based scheduling uses the reputation system to select trusted hosts [37]. The reputation system evaluates trust values of hosts according to the result of the result verification. The malicious hosts can be blacklisted.

[9]Migration can be applicable to a push model

2.5.11 Organic Grid

Organic Grid [38, 39] provides a self-organizing and distributed approach to the organization of the computation. Hosts (volunteer in this thesis) keep a list of other hosts called friends list, and build tree-based overlay network.

A user starts the computation on its host. If the host receives a request from other hosts, it distributes tasks to them as the form of a mobile agent. The requesting host becomes a child of the original host. Like this, tasks are scheduled in a distributed way. Consequently, a tree topology is constructed on the fly. An agent requests its parent for more work when it completes its task. If the parent does not have tasks, it sends a request to its parent. If a host obtains results from children or finishes its tasks, it sends them to its parent.

The tree overlay network is restructured during the computation according to the performance, that is, the rate at which a host sends a result. The hosts with high performance move towards the root of the tree. This reconstruction minimizes communication delay between the root and the best host and makes it possible to firstly allocate tasks to the best hosts.

If the parent of a host fails, the node contacts its parent's ancestor. The ancestor becomes the parents of the host and the computation resumes. Every host keeps track of the unfinished tasks of children in order to tolerate the failure of tasks. If a child requests additional tasks, unfinished tasks are resent.

2.5.12 Messor

Messor[10] [40, 41] aims to support the concurrent execution of highly-parallel and compute-intensive computations. It can self-organize overlay network for the computation by using peer-to-peer technology. Messor is composed of interconnected nests. A nest is a peer entity sharing its computational and storage resources. It can generate ants (that is, autonomous agents) that travel across the nest network. It manages its resources (that is, CPU cycles and files) and executes tasks.

Every nest can submit tasks to the nest network. The submitted tasks are scheduled to other nests in a distributed way. An ant wanders about the nest network until it encounters overloaded nest in *SerarchMax* state. An ant selects the next nest randomly or according to workload. When it finds the overloaded node, it records the identifier of this nest and changes its state into *SearchMin*. From now on, it wanders about the network, looking for a light-loaded nest. When it finds the light-loaded nest, it requests the local job manager on the nest to fetch jobs from the overloaded nest, and then changes its state back to the *SearchMax* state. Ants continuously perform the process within its time-to-live. When a task is completed, the result is sent back to the original nest. Like this, Messor constructs workload-based random

[10]Messor is built on the basis of Anthill. It is Grid computing application.

graph on the fly and achieves load balancing.

2.5.13 Paradropper

Paradropper [42, 43] is a global computing system that supports self-organizing overlay network by using peer-to-peer technologies. Peers are organized into an overlay network by using small world characteristics. A new volunteer send a message to the entry point that is randomly selected among the list of peers. The entry point in turn introduces the new peer to it neighbors. A small world graph is constructed in this manner.

Every peer maintains workload. Whenever a peer accepts a task, its workload increases by 1. Whenever a peer finishes a task and returns its result, its workload decreases by 1. A new peer has the workload 0. When the workload gets changed, a peer sends *Load Change Report* messages to its neighbors.

Tasks are distributed according to workload in a distributed way. Each peer selects the target that has the smallest workload in its neighbors. When a peer gets overloaded, it can redistribute its tasks to the network. The tasks will be accepted by light-loaded peers. Consequently, Paradropper supports load balancing. In addition, powerful peers have executed more tasks than weaker peers.

2.5.14 Condor

Condor [44, 45, 46] is a batching system for high-throughput computing on large collection of distributed resources. Condor provides a job management, scheduling, resource monitoring, and resource management, etc. Particularly, Condor aims at high-throughput computing and opportunistic computing [45]. Condor is comprised of a central manager (server in this thesis) and other resources (volunteer in this thesis). A central manager is responsible for matchmaking (scheduling) and information management about job and resources.

Condor can be used to manage dedicated clusters, or to harness wasted CPU power from otherwise idle desktop workstations within the boundaries of an organization[11]. Condor can be configured to run jobs only when the keyboard and CPU are idle. If a job is running on a workstation, when the user returns, the job migrates to a different node and resumes. In order to tolerate failures, Condor transparently takes a checkpoint and subsequently resumes the job.

Condor provides ClassAd in order to describe characteristics and requirements of both jobs and resources [45, 46]. It also provides a matchmaker for matching an job with an available resource. Condor performs matchmaking as follows. Agent (client) and resources (volunteer) advertise their ClassAds to its matchmaker. The matchmaker investigates the ClassAds, and selects job and resource pairs that sat-

[11]Condor can come under Desktop Grid because of this feature.

isfy each other's constraints and preferences. Finally, they establish a contract, and then cooperate to execute a job.

ClassAd describes the special attributes: *Requirements* and *Rank* [46]. *Requirements* attribute indicates a constraint, and *Rank* attribute measures the desirability of a match. The matchmaker first selects both job and resource to satisfy *Requirements* value. Then it chooses the one with the highest *Rank* value among compatible matches. For example, *Requirements* and *Rank* have values such as architecture, operating system, memory, disk size, load, location, etc.

Condor provides DAGMan(Directed Acyclic Graph Manager) for executing dependable jobs [46]. Condor enables preemptive-resume scheduling on dedicated compute cluster resources. It can preempt a low-priority task in order to immediately start a high-priority task.

2.5.15 Kondo et al.

Kondo et al. [47, 48, 49, 50, 51] proposes centralized scheduling mechanisms in Desktop Grid. They propose resource selection approaches for short-lived application: resource prioritization, resource exclusion, and task replication [49]. Resource prioritization is to sort hosts according to some criteria such as clock rate. Resource exclusion is to exclude some hosts according to clock or makespan predication. Task replication is to replicate tasks to multiples hosts in order to reduce the probability of tasks failure and completion delay or to schedule tasks to faster hosts.

Kondo et al. [47] propose timeout mechanism. If a server does not receive result from a host within timeout, it redistributes the task to another host. In addition, they propose a scheduling mechanism by using a buffer in order to complete applications within deadline [51].

2.6 Discussion

From the taxonomy and survey, we extract the challenging issues for Desktop Grid and a direction for Desktop Grid scheduling.

2.6.1 Challenging Issues for Desktop Grid

Desktop Grid has some characteristics such as volatility, dynamic environment, lack of trust, failure, scalability, and voluntary participation.

- ***Volatility (non-dedication)***: Since Desktop Grid is based on desktop computers, it should respect the autonomy of resource providers. In other words, volunteers can leave arbitrarily in the middle of public execution, and they are allowed to execute private execution at any time while interrupting the public execution. Accordingly, they have various volunteering time (that is, the time of donation). In addition, public execution can be stopped arbitrarily on account of unexpected leaves. Moreover, the public executions get temporarily suspended by a private execution because volunteers are not totally dedicated only to public executions. Consequently, the various occurrence rate and form of volatility directly affect the execution of tasks. A scheduler must take volatility into account in order to provide good performance and reliable computation. A scheduler should use resources as quickly as possible, when they are idle or available

- ***Dynamic environment***: Resource's owners can configure its preference and can control its machine in Desktop Grid. They can freely join and leave in the middle of the executions without any constraints. Thus, the state of system (that is, load, availability, volatility, latency, bandwidth, trust, etc.) is continuously changing over time during the public execution. A scheduler should adapt to such a dynamic environment.

- ***Lack of trust***: In Desktop Grid, anonymous nodes can participate as a resource provider. Some malicious volunteers tamper with the computation and then return corrupted results. A scheduler should guarantee the correctness of results.

- ***Failure***: In Desktop Grid, volunteers are connected through Internet, so they are exposed to crash and link failures. In addition, since volunteers are not dedicated to public execution and freely leave during public execution, the execution is delayed, blocked, and even lost. A scheduler should tolerate the failures and volatility.

- ***Heterogeneity***: Desktop Grid is based on desktop computers at the edge of Internet. Volunteers have heterogeneous properties such as CPU, memory, network bandwidth, latency. In addition, each volunteer has a various occurrence rate of failures and volatility, availability, and trust according to its execution behavior. The heterogeneity delays the overall completion time or makes scheduling decision more difficult.

- ***Scalability***: Centralized scheduling has some drawbacks such as scalability, single point of failure, etc. Particularly, a central server suffers from overhead to perform scheduling and to manage various volunteers and jobs when the number of volunteers increases. On the contrary, distributed scheduling provides scalability because scheduling decision is made at each volunteer. However, distributed scheduling has some drawbacks such as performance as compared with centralized scheduling because it conducts scheduling on the basis of local and partial information without a global view.

- ***Voluntary participation***: In Desktop Grid, resource providers are mainly voluntary participants without any reward for their donation of resources. In order to encourage resource providers to reliably and eagerly donate their resources for a long time, a scheduler should consider reputation and incentive mechanisms.

2.6.2 A Direction for Desktop Grid Scheduling

To overcome the above challenges, Desktop Grid scheduling should deal with the following considerations.

- ***Coupling resource grouping with scheduling***: In Desktop Grid, volunteers should be grouped together according to their properties (such as capability, performance, availability, workload, reputation/trust, volatility, etc.) in order to execute tasks and manage volunteers efficiently. The resource grouping method creates computational overlay network (CON). The CON makes an effect on relieving resource's heterogeneity. In addition, a scheduler can expand centralized or distributed approaches to hierarchical approach by constructing multiple CONs and applying different scheduling algorithms to each CON [54, 55, 56]. Thus, it is very important how to make the CON depending on volunteer's properties, because scheduling, resource management, and information management are performed on the basis of characteristics or topology of CON. However, existing centralized DG systems [14, 18, 22, 31] do not provide how to construct the CON on the basis of volunteering time, volatility, reputation/trust, and credibility. They simply construct a CON depending on resource's basic properties (for example, disk space, or OS type). On the other hand, existing decentralized Desktop Grid systems [25, 34, 38, 41, 42] provide how to construct a CON depending on registration time, timezone, performance, or workload, but they do not consider volatility, volunteering time, credibility, and reputation/trust, which directly affect reliability, completion time, and result correctness. Moreover, resource grouping is not tightly related with scheduling (especially, result certification, replication, and reassignment). As a result, uncoupling between resource grouping and scheduling causes a lot of overhead and degrades performance. A new scheduler in Desktop Grid should provide more delicate construction methods of CON and couple resource grouping with scheduling.

- ***Reputation or Incentive-based scheduling***: In Desktop Grid, volunteers can be eager, reliable, volatile, selfish, or malicious. In other words, volunteers have various volunteering time, volatility, credibility, and availability according to their execution behavior. In order to score and rank the volunteers, and then reward or punish them according to the assessment, a reputation system should be coupled with scheduling [80, 81, 82]. A reputation-based scheduling can choose high qualified resources, so that it can improve the reliability and performance. An incentive-based scheduling focuses on punishing (for example, exclusion) volatile, selfish, or malicious volunteers. Thus, it gives an incentive to eager and reliable volunteers. A new scheduler of Desktop Grid should consider reputation and incentive-based scheduling.

- ***Scheduling for Result certification***: In Desktop Grid, some volunteers may behave erratically or maliciously. In other words, some malicious volunteers may tamper with the computation and return corrupted results. In addition, a variety of hardware and software malfunction leads to variations in numerical processing [15]. Therefore, Desktop Grid needs to detect and tolerate the erroneous result in order to guarantee a reliable execution in such a distrusted environment. To this end, Desktop Grid exploits result certification mechanisms such as majority voting and spot-checking [15, 23, 37, 56, 98]. Result certification should be tightly related with scheduling in the sense that both the special task for spot-checking and the redundant tasks for voting are allocated to volunteers in a scheduling procedure. However, existing Desktop Grid systems simply use eager scheduling, so they have a lot of problems because the eager scheduling does not consider the properties of volunteers such as volatility, volunteering time, and credibility during result certification. As a result, there are high overhead, performance degradation, and scalability problems. Desktop Grid should provide a new scheduling mechanism for result certification considering resource's properties.

- ***Dynamic, adaptive, or fault tolerant scheduling***: Desktop Grid is based on desktop computers at the edge of Internet, so volunteers can freely join and leave in the middle of the public execution without any constraints, and they are exposed to crash and link failures. Moreover, the resource's properties (workload, bandwidth, availability, volatility, etc.) are changing over time. Desktop Grid should adapt to dynamically changing environment. In other words, Desktop Grid scheduling should be able to obtain dynamically changing state and then consider them as environmental inputs or stimuli when making decisions. Particularly, a Desktop Grid scheduler should be able to deal with volatility and failures that occur frequently in a dynamic environment, in order to provide reliability.

- ***Distributed scheduling***: Distributed DG [26, 35, 38, 41, 43] mainly uses distributed scheduling because there is no central server. However, existing distributed scheduling mechanisms allocate tasks to volunteers according to workload or performance as well as randomly. They do not use volatility, volunteering time, credibility, and reputation/trust, which directly affect reliability, completion time, and result correctness. Moreover, they do not provide replication or result certification mechanism, which is necessary in Desktop Grid. A new distributed scheduler should consider these properties and should provide scheduling algorithm coupled with replication and result certification.

2.7 Related Work

2.7.1 Taxonomy and Survey of Grid and Desktop Grid

There are several taxonomies and survey of Grid. Baker et al. [4] attempted to present the state-of-the-art of Grid computing and survey emerging Grid computing projects. Krauter et al. [69] proposed the taxonomy of Grid focusing on resource management. Venugopal et al. [59] proposed the taxonomy of Data Grids according to organization, data transport, data replication, and scheduling. Foster et al. [5] compared P2P and Grid computing.

Sarmenta [24] classified volunteer computing into application-based and web-based (java-based). Chien et al. [21] classified Desktop Grid into Internet Grid and Enterprise Grid. However, they do not provide a mapping of taxonomy to the existing Desktop Grid systems. Moreover, the taxonomy we propose in this thesis is more delicate and expanded.

2.7.2 Taxonomy and Survey of Scheduling

There are several proposed taxonomy for scheduling in distributed, heterogeneous computing, and Grid computing environment.

Casavant et al. [61] proposed the taxonomy of scheduling in general-purpose distributed computing systems. They classified scheduling into local and global scheduling, and then classified global scheduling into static and dynamic according to the time of decision making. Rotithor

[62] proposed the taxonomy of dynamic scheduling in distributed computing systems according to state estimation and decision making.

Braun et al. [63] proposed the taxonomy of heterogeneous computing systems. The taxonomy is defined in three major parts: application model, platform model, mapping strategy characterization. Ali et al. [64] characterized resource allocation heuristics for heterogeneous computing systems according to workload, platform, and mapping strategy[12]. Ekmecic et al. [65] modified the taxonomy proposed by Casavant et al. [61]. Maheswaran et al. [67] proposed various online (MCT, MET, SA, KPB, OLB) and batch mode heuristics (min-min, max-min, sufferage) for dynamic scheduling in a heterogeneous computing environment. Braun et al. [68] proposed the comparison of static scheduling heuristics (OLB, MET, MCT, min-min, max-min, duplex, GA, SA, GSA, Tabu, A*) for heterogeneous distributed computing systems.

Krauter et al. [69] proposed a taxonomy and survey of grid resource management systems. The taxonomy is defined in four aspects: scheduling organization, state estimation, rescheduling, and scheduling policy. Yu et al. [70] proposed the taxonomy of scientific workflow scheduling for Grid computing. The taxonomy is defined in four aspects: archi-

[12]The three categories of our taxonomy proposed in this thesis (that is, application, resource, and scheduler's perspective) is similar to the three categories of the taxonomy proposed by Barun et al [63] and Ali et al. [64]. However, our taxonomy is Desktop Grid-oriented. It also considers the new and delicate items for Desktop Grid scheduling such as dedication, volatility, resource grouping, result certification, opportunism, reputation/incentive, etc.

tecture, decision making, planning scheme, and scheduling strategies. Yeo et al. [71] proposed the taxonomy of market-based resource management system for utility-driven cluster computing. They proposed a taxonomy of job model (processing type, composition, QoS specification, QoS update) and resource allocation model (domain, update, and QoS support). Venugopal et al. [59] proposed the taxonomy of scheduling for Data Grid according to application model, scope, data replication, utility function, and locality. Hamscher et al. [72] proposed centralized, hierarchical, and decentralized scheduling architectures for Grid.

There are several taxonomy and survey of scheduling for heterogeneous computing and Grid. However, there is no taxonomy and survey of scheduling for Desktop Grid. Particularly, our taxonomy deals with volunteer's key properties (such as volatility, dedication, reputation/trust, etc.) in a Desktop Grid environment and considers the resource grouping (construction of computational overlay network), result certification, and reputation/incentive scheduling aspects.

Chapter 3

System Model

We illustrate a new execution model and a failure model defined in this thesis.

3.1 Execution Model

This thesis considers centralized DG and volunteer DG computing environment. In such an environment, we propose a new execution model. The execution model consists of eight phases: registration, job submission, resource grouping, task allocation, task execution, task result return, result certification, and job result return phase as shown in the Figure 3.1. We newly add a resource grouping phase to existing execution model. We also modify the existing execution model as follows. The rest of the phases are the same as existing centralized DG.

In the registration phase, volunteers register basic properties such as CPU, memory, OS as well as additional properties such as , volunteering time, availability, credibility, etc. The additional properties reflect

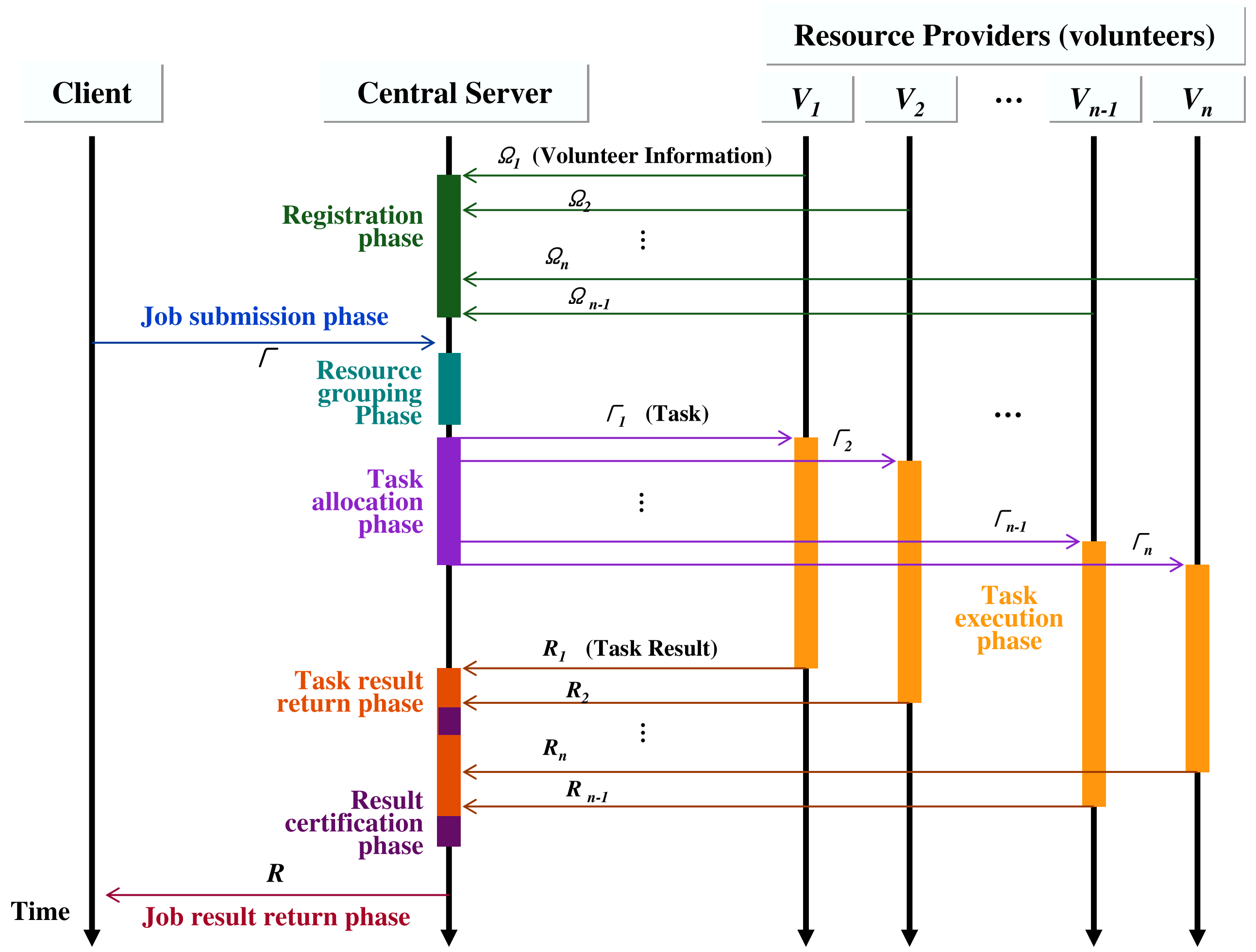


Figure 3.1: New execution model of Desktop Grid

dynamical Desktop Grid computing.

In the resource grouping phase, a server constructs groups according to capability, volatility, availability, reputation, and trust of volunteers. Scheduling is performed, depending on the characteristics of groups.

In the task allocation phase, a server allocates tasks to volunteers on a group basis. In other words, it applies scheduling, replication, and result certification algorithms to each group. In the presence of failures, a server reschedules the failed tasks according to fault tolerant algorithm.

In the result certification phase, a server checks the correctness of the returned results on the basis of group.

3.2 Failure Model

In a Desktop Grid computing environment, volunteers are connected through the Internet, and therefore are exposed to crash and link failures. In addition, since Desktop Grid computing is based on voluntary participants, the autonomy of volunteers is respected. In other words, volunteers can leave arbitrarily in the middle of public execution and are allowed to interrupt public execution at any time. In a Desktop Grid computing environment, volunteer autonomy failures occur much more frequently than crash and link failures. Therefore, volunteer autonomy failures should specially be dealt with, while they are distinguished from traditional failures. Moreover, volunteers have various occurrence rates and types of volunteer autonomy failures. Since the heterogeneous occurrence rates and types of volunteer autonomy failures affect computation directly, a scheduling mechanism must take them into account in order to obtain better performance and guarantee reliable computation. To this end, volunteer autonomy failures are first defined conceptually [54, 58].

In order to clarify definition of volunteer autonomy failures, the notations in Table 3.1 are used. First, the join and leave patterns of a volunteer are categorized. The patterns are categorized into expected join (EJ), expected leave (EL), unexpected join (UJ), and unexpected leave (UL).

$$\begin{aligned}
EJ &\triangleq (T[_{V_i \curvearrowright \xi_i}] = V_i.\Upsilon_s t) \\
EL &\triangleq (T[_{V_i \curvearrowleft \xi_i}] = V_i.\Upsilon_{tt}) \\
UJ &\triangleq ((T[_{V_i \curvearrowright \xi_i}] \neq V_i.\Upsilon_{st}) \\
UL &\triangleq (T[_{V_i \curvearrowright \xi_i}] \neq V_i.\Upsilon_{tt})
\end{aligned}$$

UJ is categorized into before-unexpected-join UJ^b, middle-unexpected-join UJ^m, and after-unexpected-join UJ^a. In addition, unexpected-leave UL is categorized into before-unexpected-leave UL^b, middle-unexpected-leave UL^m, and after-unexpected-leave UL^a.

$$\begin{aligned}
&UJ = \{UJ^b,\ UJ^m,\ UJ^a\} \\
&UJ^b \triangleq (T[_{V_i \curvearrowright \xi_i}] < V_i.\Upsilon_{st}) \\
&UJ^m \triangleq (V_i.\Upsilon_{st} < T[_{V_i \curvearrowright \xi_i}] < V_i.\Upsilon_{tt}) \\
&UJ^a \triangleq (V_i.\Upsilon_{tt} < T[_{V_i \curvearrowright \xi_i}])
\end{aligned}$$

$$\begin{aligned}
&UL = \{UL^b,\ UL^m,\ UL^a\} \\
&UL^b \triangleq (T[_{V_i \curvearrowleft \xi_i}] < V_i.\Upsilon_{st}) \\
&UL^m \triangleq (V_i.\Upsilon_{st} < T[_{V_i \curvearrowleft \xi_i}] < V_i.\Upsilon_{tt}) \\
&UL^a \triangleq (V_i.\Upsilon_{tt} < T[_{V_i \curvearrowleft \xi_i}])
\end{aligned}$$

Table 3.1: Notations for volunteer autonomy failures

V_i	A Volunteer ($0 \leq i \leq n$)
Γ_m	A task executed by a volunteer
ξ_i	Public execution of a task Γ_m at V_i
I_{ξ_i}	Time interval of public execution ξ_i
Υ	Volunteering time that is the period when a volunteer is supposed to provide its resources
Υ_{st}	The start time when a volunteer V_i is supposed to provide its resources
Υ_{tt}	The termination time when a volunteer V_i is supposed to provide its resources
$V_i \curvearrowright \xi_i$	The join event, that is, a volunteer V_i participates in public execution ξ_i
$V_i \curvearrowleft \xi_i$	The leave event, that is, a volunteer V_i leaves public execution ξ_i
$\mathrm{T}[V_i \curvearrowright \xi_i]$	The time when $V_i \curvearrowright \xi_i$ happens
Π_i	An individual job that is performed by a personal user at V_i
π_i	Private execution of a individual job Π_i
$\triangleq$	The symbol means ”occurs when”

Volunteer autonomy failures (Λ) are classified into volunteer volatility failure (Φ) and volunteer interference failure (Ψ).

$$\Lambda = \{\Phi,\ \Psi\}$$

Definition 1 (Volunteer volatility failure) *Volunteer volatility failure Φ is abortion of public execution that is caused by freely leaving of the public execution ξ_i of a task Γ_i.*

$$\Phi \ \triangleq\ T[_{V_i \curvearrowleft \xi_i}] \in I_{\xi_i}$$

The volunteer volatility failure is categorized as follows: unexpected-before Φ^b, unexpected-middle Φ^m, expected Φ^e, and unexpected-after Φ^a.

$$\begin{aligned}
&\Phi = \{\Phi^b,\ \Phi^m,\ \Phi^e,\ \Phi^a\} \\
&\Phi^b \ \triangleq\ (T[_{V_i \curvearrowleft \xi_i}] \in I_{\xi_i}) \vee (T[_{V_i \curvearrowleft \xi_i}] < V_i.\Upsilon_{st}) \\
&\Phi^m \ \triangleq\ (T[_{V_i \curvearrowleft \xi_i}] \in I_{\xi_i}) \vee (V_i.\Upsilon_{st} < T[_{V_i \curvearrowleft \xi_i}] < V_i.\Upsilon_{tt}) \\
&\Phi^e \ \triangleq\ (T[_{V_i \curvearrowleft \xi_i}] \in I_{\xi_i}) \vee (T[_{V_i \curvearrowleft \xi_i}] = V_i.\Upsilon_{st}) \\
&\Phi^a \ \triangleq\ (T[_{V_i \curvearrowleft \xi_i}] \in I_{\xi_i}) \vee (V_i.\Upsilon_{tt} < T[_{V_i \curvearrowleft \xi_i}])
\end{aligned}$$

Definition 2 (Volunteer interference failure) *Volunteer interference failure Ψ is temporary suspension of public execution ξ_i that is caused by private execution π_i of an individual job Π_i.*

$$\Psi \ \triangleq\ (T[\pi_i] \in I_{\xi_i})$$

Volunteer interference failure Ψ is categorized into expected Ψ_{ei} and unexpected Ψ_{ui}. Ψ_{ei} occurs when private execution interferes with public execution regularly (for example, reserved virus checking), but Ψ_{ui} occurs when private execution that starts from keyboard or mouse movement interferes with public execution irregularly (for example, temporary email checking etc.).

Φ and Ψ are different from crash failure in that the operating system is alive in the presence of Φ and Ψ, whereas it shuts down in the presence of crash failure [58, 99, 100]. Φ is different from crash failure in that Φ occurs by the will of volunteers [85, 99, 100]. Ψ is different from Φ in that a Desktop Grid computing system is alive in the presence of Ψ, whereas it is not operating in the case of Φ.

Φ is related to the completion of public execution. For example, if a leave event arbitrarily happens in the middle of public execution, this execution is stopped (or aborted). As a result, the execution is not completed. That is, Φ hinders the completion of execution. On the other hand, Ψ is related to the continuity of public execution. For example, if a personal user frequently performs private execution during public execution, public execution is temporarily suspended. Consequently, the public execution cannot proceed continuously. That is, Ψ obstructs the continuity of execution.

Chapter 4

Group-based Adaptive Scheduling Mechanism

This chapter introduces a resource grouping method, which classifies and constructs various groups according to volunteer's properties such as dedication, volatility, availability, credibility, etc. Then, it proposes group-based adaptive scheduling mechanisms, which apply scheduling, replication, result certification, and fault tolerant algorithms to each group.

4.1 Resource Grouping Method

Resource grouping provides a method of forming volunteer groups. Volunteer group is a set of volunteers that have similar properties such as dedication, volatility, failure, and trust. In order to apply different scheduling mechanisms suitable for the properties of volunteers in a scheduling procedure, volunteers are required to first be formed into homogeneous groups. First, we classify volunteers according to their properties. Then, we construct and characterize volunteer groups.

4.1.1 Criteria for Resource Grouping

When volunteers are classified, their CPU, memory, storage, and network capacities are important factors. The most important factors, however, are location, volunteering time, volunteer autonomy failures, volunteer availability, and volunteer credibility in the sense that the completion and continuity of computation, the reliability and correctness of results, and performance are tightly related with dedication, volatility, failure, and trust (see Figure 4.1). In a Desktop Grid computing environment, the capacities of desktop computers are very heterogeneous, and the degree of volatility, dedication, and trust fluctuate considerably during execution [23, 47, 48, 53, 54].

Volatility and Volunteer Availability.

Volatility means that volunteers can leave in the middle of public execution. Volunteers also are exposed to crash and link failures because they are connected through Internet. Volatility and failures lead to the delay and blocking of the execution of tasks and even partial or entire loss of the executions. Thus, they are tightly related with reliability of computation. We newly define *volunteer availability* to express volatility and failures as follows:

Definition 3 (Volunteer availability) *Volunteer availability (α_v) is*

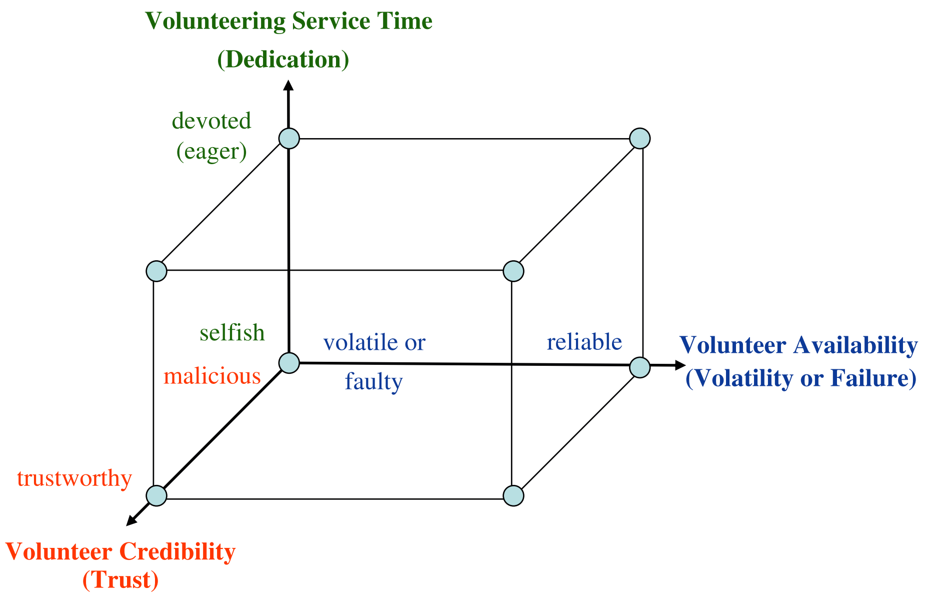


Figure 4.1: Criteria for resource grouping

the probability that a volunteer will be correctly operational and be able to perform public execution.

$$\alpha_v = \frac{MTTVAF}{MTTVAF + MTTR}$$

Here, the $MTTVAF$ represents "mean time to volunteer autonomy failures" and the $MTTR$ represents "mean time to rejoin". The $MTTVAF$ represents the average time before the volunteer autonomy failures happen, and the $MTTR$ means the mean duration of volunteer autonomy failures. The α_v reflects the degree of volunteer autonomy failures, whereas the traditional availability in distributed systems is mainly related with the crash failure [99, 100].

$MTTVAF$ and $MTTR$ are recalculated dynamically when a volunteer detects Φ and Ψ. Here, MVT represents "mean volunteering time". The symbol $\bowtie$ represents a combination of the two events. The symbol $\biguplus$ represents the union of time intervals. The parameter μ is a weight constant. When a volunteer executes a task, the μ is initially set to 1. The μ increases whenever Φ and Ψ occur. The μ is reset to 1 when the volunteer finishes its task.

Case 1: $UJ^b, \quad \Phi^b, \quad or \quad \Phi^a$

$$MTTVAF = MTTVAF + \mu \times \frac{\{I_{(UJ^b \bowtie\ EJ)} \biguplus I_{(UJ^b \bowtie\ \Phi^b)} \biguplus I_{(EL \bowtie\ \Phi^a)}\}}{MVT}$$

$$MTTR = MTTR - \mu \times \frac{\{I_{(UJ^b \bowtie\ EJ)} \biguplus I_{(UJ^b \bowtie\ \Phi^b)} \biguplus I_{(EL \bowtie\ \Phi^a)}\}}{MVT}$$

$$MVT = MVT + \mu \times \frac{\{I_{(UJ^b \bowtie\ EJ)} \biguplus I_{(UJ^b \bowtie\ \Phi^b)} \biguplus I_{(EL \bowtie\ \Phi^a)}\}}{MVT}$$

Case 2: $UJ^m \quad or \quad \Phi^m$

$$MTTVAF = MTTVAF - \mu \times \frac{\{I_{(EJ \bowtie\ UJ^m)} \biguplus I_{(\Phi^m \bowtie EL)}\}}{MVT}$$

$$MTTR = MTTR + \mu \times \frac{\{I_{(\Phi^m \bowtie\ UJ^m)}\}}{MVT}$$

$$MVT = MVT - \mu \times \frac{\{I_{(EJ \bowtie\ UJ^m)} \biguplus I_{(\Phi^m \bowtie EL)}\}}{MVT}$$

Case 3: $\Psi_{ei} \quad or \quad \Psi_{ui}$

$$MTTVAF = MTTVAF - \mu \times \frac{\{I_{\Psi_{ei}} \uplus I_{\Psi_{ui}}\}}{MVT}$$

$$MTTR = MTTR + \mu \times \frac{\{I_{\Psi_{ei}} \uplus I_{\Psi_{ui}}\}}{MVT}$$

$$MVT = MVT - \mu \times \frac{\{I_{\Psi_{ei}} \uplus I_{\Psi_{ui}}\}}{MVT}$$

Cases 1 and 2 describe how to calculate volunteer availability in the case of volunteer volatility failure and unexpected-join. Case 3 describes how to calculate volunteer availability when volunteer interference failure occurs. The parameter μ is used in order to reflect the rate and frequency of volunteer autonomy failures into volunteer availability. For example, if volunteer autonomy failures occur repeatedly and frequently, volunteer availability drops rapidly. Moreover, the mean volunteering time affects the volunteer availability. For example, if the mean volunteering time is short, volunteer availability is considerably affected by volunteer autonomy failures. In Case 1, volunteer availability increases because unexpected volunteering time is additionally provided. Conversely, in Cases 2 and 3, volunteer availability decreases because volunteering time decreases due to volunteer autonomy failures.

Dedication and Volunteer Service Time.

Dedication means that how much volunteers participate in public execution. Dedication is related with donation (or participation) time.

A volunteer can be eager or selfish according to degree of dedication. We define *volunteering time* and *volunteer service time* to express the degree of dedication as follows:

Definition 4 (Volunteering time) *Volunteering time (Υ) is the period when a volunteer is supposed to donate its resources.*

$$\Upsilon = \Upsilon_R + \Upsilon_S$$

Here, the *reserved volunteering time* (Υ_R) represents the reserved time when a volunteer provides computing resources. A volunteer mostly performs public execution during Υ_R, rarely performing private execution. However, the *selfish volunteering time* (Υ_S) represents unexpected volunteering time. Thus, a volunteer usually performs private execution during the Υ_S, and sometimes performs public execution.

Volunteer service time is defined as follows:

Definition 5 (Volunteering service time) *Volunteering service time (Θ) is the expected service time when a volunteer participates in the public execution during Υ.*

$$\Theta = \Upsilon \times \alpha_v$$

In a scheduling procedure, Θ is more appropriate than Υ because Θ represents the time when a volunteer actually executes each task in the presence of volunteer autonomy failures Λ. Therefore, volunteer groups are constructed according to Θ.

Trust and Volunteer Credibility.

Trust means that volunteers execute tasks correctly and return the correct result. Malicious volunteers may tamper with the overall computation. We define *volunteer credibility* to express the trust as follows:

Definition 6 (Volunteer credibility) *Volunteer credibility C_v is the probability that the result produced by a volunteer is correct.*

$$C_v = \frac{CR}{ER + CR + IR}$$

Here, *ER* represents the number of erroneous results, *CR* represents the number of correct results, and *IR* represents the number of incomplete results. $ER + CR + IR$ means the total number of tasks that a volunteer executes. The IR occurs when a volunteer does not complete spot-checking or majority voting on account of crash failure and volunteer autonomy failures. If a volunteer passes the spot-checking, the credibility becomes higher. If volunteers within voting group reach the agreement for majority voting, their credibility also becomes higher.

4.1.2 Constructing and Characterizing Volunteer Groups

Classification I according to α_v and Θ.

Volunteers are categorized into four classes according to α_v and Θ.

Volunteers are first categorized into region volunteers or home volunteers according to their location. *Home volunteers* are defined as resource donators at home. *Region volunteers* are a set of resource donators that are generally affiliated with organizations including universities, institutions, and so on. Region volunteers are connected to LAN or Intranet, whereas home volunteers are connected to the Internet.

Volunteers are categorized into four classes according to Υ and α_v (see Figure 4.2 (a)). The class A is a set of volunteers that have long Υ and high α_v. The class B is a set of volunteers that have short Υ and high α_v. The class C is a set of volunteers that have long Υ and low α_v. The class D is a set of volunteers that have short Υ and low α_v.

A server selects volunteers as volunteer group members according to the properties of volunteers such as location, volunteer availability, and volunteering service time.

If volunteer groups are constructed on the basis of location, region volunteers belong to the same group, and home volunteers are formed into the same group in order to reduce the communication cost between members.

When both α_v and Θ are considered in grouping the volunteers, the volunteer groups are categorized into four classes (see Figure 4.2 (b)). Here, Δ is the expected computation time of a task.

Volunteers are classified into four classes: A', B', C', and D' volunteer groups. If volunteers have a high α_v and $\Theta \geq \Delta$, they are included in the class A'. If volunteers have a high α_v and $\Theta < \Delta$, they are included in the class B'. If volunteers have a low α_v and $\Theta \geq \Delta$, they are included in the class C'. If volunteers have a low α_v and $\Theta < \Delta$, they

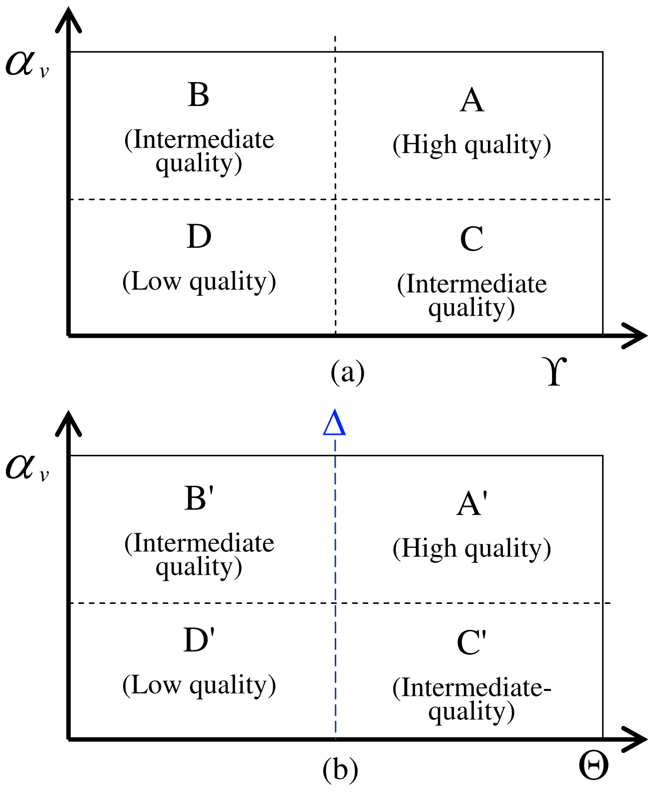


Figure 4.2: Volunteer groups according to α_v and Θ

are included in the class D'.

Volunteer groups are constructed using the algorithm of volunteer group construction (see Figure 4.3).

1) *The registered volunteers are classified into home or region volunteers, depending on their location.*

2) *The home and region volunteers are classified into A, B, C, and D classes by volunteering time and volunteer availability, respectively.*

3) *The volunteer groups are constructed according to volunteering service*

```
// To classify the registered volunteers($V$) into home or region volunteers
ClassifyVolunteersByLocation($V$);
// To classify the home and region volunteers into $A$, $B$, $C$, $D$ classes, respectively
ClassifyVolunteers($V$);
// To construct volunteer groups
if ($V_i.\Theta \geq \Delta$) then   // $V_i$ : one of the classified volunteers
    if ($V_i \in V_A || V_i \in V_B$) then   // $V_A$ : $A$ class, $V_B$: $B$ class
        $V_i \rightarrow VG_{A'}$;   // $\rightarrow$ : assign, $VG_{A'}$ : $A'$ volunteer group
    else // $V_i \in V_C || V_i \in V_D$, here, $V_C$ : $C$ class, $V_D$: $D$ class
        $V_i \rightarrow VG_{C'}$;   // $VG_{C'}$ : $C'$ volunteer group
    fi;
else   // $V_i.\Theta < \Delta$
    if ($V_i \in V_A || V_i \in V_B$) then
        $V_i \rightarrow VG_{B'}$;   // $VG_{B'}$ : $B'$ volunteer group
    else
        $V_i \rightarrow VG_{D'}$;   // $VG_{D'}$ : $D'$ volunteer group
    fi;
fi;
```

Figure 4.3: Algorithm of volunteer group construction according to α_v and Θ

time and volunteer availability.

The volunteer groups have the following properties. The A' volunteer group has a high Θ and high α_v sufficient to reliably execute tasks. It is used as deputy volunteers that host the scheduling mobile agents. The B' volunteer group has a high α_v, but low Θ. It cannot complete their tasks because of lack of computation time. The C' volunteer group has a high Θ, but low α_v. It has the time enough to execute tasks. However, volunteer autonomy failures occur frequently during execution. Therefore, it requires fault tolerant mechanism to execute tasks reliably. The D' volunteer group has a low Θ and low α_v. It has insufficient time to execute tasks. Moreover, volunteer autonomy failures occur frequently in the middle of execution. Among the volunteer groups, the A' and C' volunteer groups mainly execute tasks because of sufficient time. If a task migrates during execution, the B' volunteer group can be used as migration places when the A' and C' volunteer groups suffer from failures. Otherwise, the B' volunteer group is not appropriate to distribute tasks because its volunteering service time is too short to complete a task. In this case, it executes tasks for testing, that is, to measure its properties. The D' volunteer group gives rise to a high management cost due to lack of time as well as low volunteer availability. The D' volunteer group also only executes tasks for testing. If checkpointing is used, the B' and D' volunteer groups can be used to execute non-time-critical applications.

Classification II according to α_v, Θ, and C_v.

Volunteers are categorized into four classes according to α_v, Θ, and C_v.

First, the registered volunteers are classified into A', B', C', and D' classes depending on volunteering service time and volunteer availability as shown in Figure 4.4 (a). Then, the classified volunteers are classified

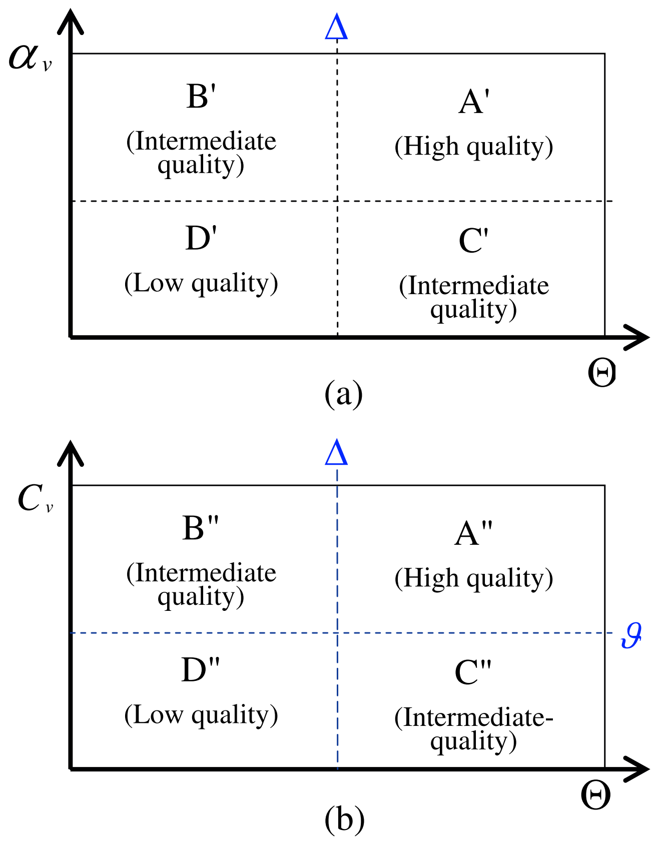


Figure 4.4: Volunteer groups according to α_v, Θ, and C_v

into each volunteer group according to volunteer credibility. Volunteer groups are categorized into four classes (A'', B'', C'', and D'' classes) as shown in Figure 4.4 (b). Here, Δ is the expected computation time of a task. ϑ is the desired credibility threshold.

Volunteer groups are constructed by the algorithm of volunteer group construction as shown in Figure 4.5.

The A'' volunteer group has both high C_v, high Θ, and high α_v enough to execute tasks reliably. There is high possibility to produce

```
// $V_A$ : $A'$ class, $V_B$: $B'$ class, $V_C$ : $C'$ class, $V_D$: $D'$ class
// $VG_{A''}$ : $A''$ class, $VG_{B''}$: $B''$ class, $VG_{C''}$ : $C''$ class, $VG_{D''}$: $D''$ class
// To classify the registered volunteers into $A'$, $B'$, $C'$, $D'$ classes, respectively
ClassifyVolunteers($V$);
// To construct volunteer groups
**if** ($V_i \in V_A$) **then**   // $V_i$ : one of the classified volunteers
    **if** ($V_i.C_v \geq \vartheta$) **then**   // $V_i.C_v$ : $C_v$ of $V_i$
        $V_i \rightarrow VG_{A''}$;   // $\rightarrow$ : assign
    **else**
        $V_i \rightarrow VG_{C''}$;
    **fi**;
**else if** ($V_i \in V_B$) **then**
    **if** ($V_i.C_v \geq \vartheta$) **then**
        $V_i \rightarrow VG_{B''}$;
    **else**
        $V_i \rightarrow VG_{D''}$;
    **fi**;
**else if** ($V_i \in V_C$) **then**
    $V_i \rightarrow VG_{C''}$;
**else**
    $V_i \rightarrow VG_{D''}$;
**fi**;
```

Figure 4.5: Algorithm of volunteer group construction according to α_v, Θ, and C_v

correct results in the A'' volunteer group. The B'' volunteer group has high C_v and high α_v, but low Θ. It has a high possibility to produce correct results. However, it cannot complete their tasks because of lack of the execution time. In addition, volunteer autonomy failures occur frequently in the middle of execution. The C'' volunteer group has high Θ, but low C_v and low α_v. It has time enough to execute tasks. However, its results might be incorrect. Therefore, in order to strength the credibility, C'' volunteer group must do more spot-checking or placing more redundancy for voting than A'' or B'' volunteer group. The D'' volunteer group has low C_v, low Θ, and low α_v. It has no time enough to execute tasks. In addition, there is scarcely any possibility to produce correct results. Moreover, volunteer autonomy failures occur frequently in the middle of execution. Therefore, tasks are not allocated to the D'' volunteer group, because not only management cost is too expensive, but also results are incorrect.

4.1.3 Maintaining Volunteer Groups

The volunteer groups are maintained by three mode: task-based, time-based, and count-based modes. In the *task-based mode*, whenever a task is completed, volunteer groups are built. The *time-based mode* builds volunteer groups at the regular intervals if the tasks to schedule remain. The *count-based mode* constructs volunteers groups when the number of participating volunteers is larger than or equal to a predefined number k. The k depends on the size of volunteer groups or the number of redundancy. The size of a volunteer group is mainly related with the maintenance cost (that is, the scheduling and management cost of task mobile agents, fault tolerance, replication, etc.). The volunteer groups are kept until the scheduling agent cannot further distribute tasks to members. For example, if all members have insufficient time to execute a task, volunteer groups are dismissed. The members of volunteer groups are partially replaced by others if a volunteer fails.

4.2 Agent-based Group Scheduling

4.2.1 Why Agent?

In existing Desktop Grid computing system, a server suffers from high overhead. A server maintains properties of volunteers such as CPU, memory, OS, location (address), and so on. According to the properties, the server has responsibility for scheduling in a centralized way. In addition, the server performs the fault tolerant mechanism if volunteers fail. Since a scheduling mechanism is performed only by the server, various scheduling mechanisms are not performed at a time according to volunteer properties. To solve these problems, we make use of mobile agent technology.

Mobile agent is a software program that migrates from one node to another while performing some tasks on behalf of a user [87, 88, 89, 90, 91]. Mobile agent has some benefits as follows [87, 88, 89, 90, 91].

1) A mobile agent can reduce network load and latency by dispatching

the agents to a remote node which has the required services or data and then executing it locally at the host.
2) A mobile agent can solve frequent and intermittent disconnection since it executes some tasks asynchronously and autonomously. Once a mobile agent is dispatched to a destination node, it does not require direct connection with a user any more. Because a mobile agent is performed asynchronously and autonomously on behalf of a user even though a user (that is, mobile device) is disconnected from the network.
3) A mobile agent enables dynamic service customization and software deployment since a mobile agent encapsulates some services or protocols in mobility entity.
4) A mobile agent can adapt to heterogeneous environment and dynamic changes because it is computer-independent and transport-independent and also reacts autonomously according to execution environment.

There are some advantages to make use of mobile agent in Desktop Grid computing environment.
1) ***Several scheduling algorithms cab be performed at a time according to the properties of volunteers.*** For example, various scheduling mechanisms can be implemented as mobile agents. If there are various volunteer groups, the best appropriate scheduling mobile agent is assigned to the volunteer group according to the property of volunteer group. Existing Desktop Grid computing system, however, cannot apply various scheduling mechanisms at a time because one scheduling mechanism is performed only by a server.

2) ***A mobile agent can decrease the overhead of server by performing scheduling and fault tolerance procedures in a decentralized way.*** The scheduling mobile agents are distributed to volunteer groups. After that, they perform each scheduling and fault tolerance procedure in each volunteer group. Therefore, the server does not undergo the overhead any more.

3) ***A mobile agent can adapt to a dynamical Desktop Grid computing environment.*** In a Desktop Grid computing environment, volunteers can join and leave at any time. A mobile agent can tolerate the volunteer autonomy failures by using migration and replication functionalities which the mobile agent itself provides.

4.2.2 Agent based Desktop Grid Computing Model

A mobile agent can adapt to dynamic environmental changes as well as various properties of volunteers. In addition, since a mobile agents are executed in a distributed way, it can decrease the overhead of a server.

The mobile agent based Desktop Grid computing works like the execution model of existing Desktop Grid computing. Several phases, however, works differently, as shown in Figure 4.6.

In the registration phase, volunteers register basic properties such as CPU, memory, OS as well as additional properties such as , *volunteering time*, *volunteering service time*, *volunteer availability*, *volunteer credibility* etc. Since the additional properties reflect dynamical Desktop Grid computing, they are more important than basic properties.

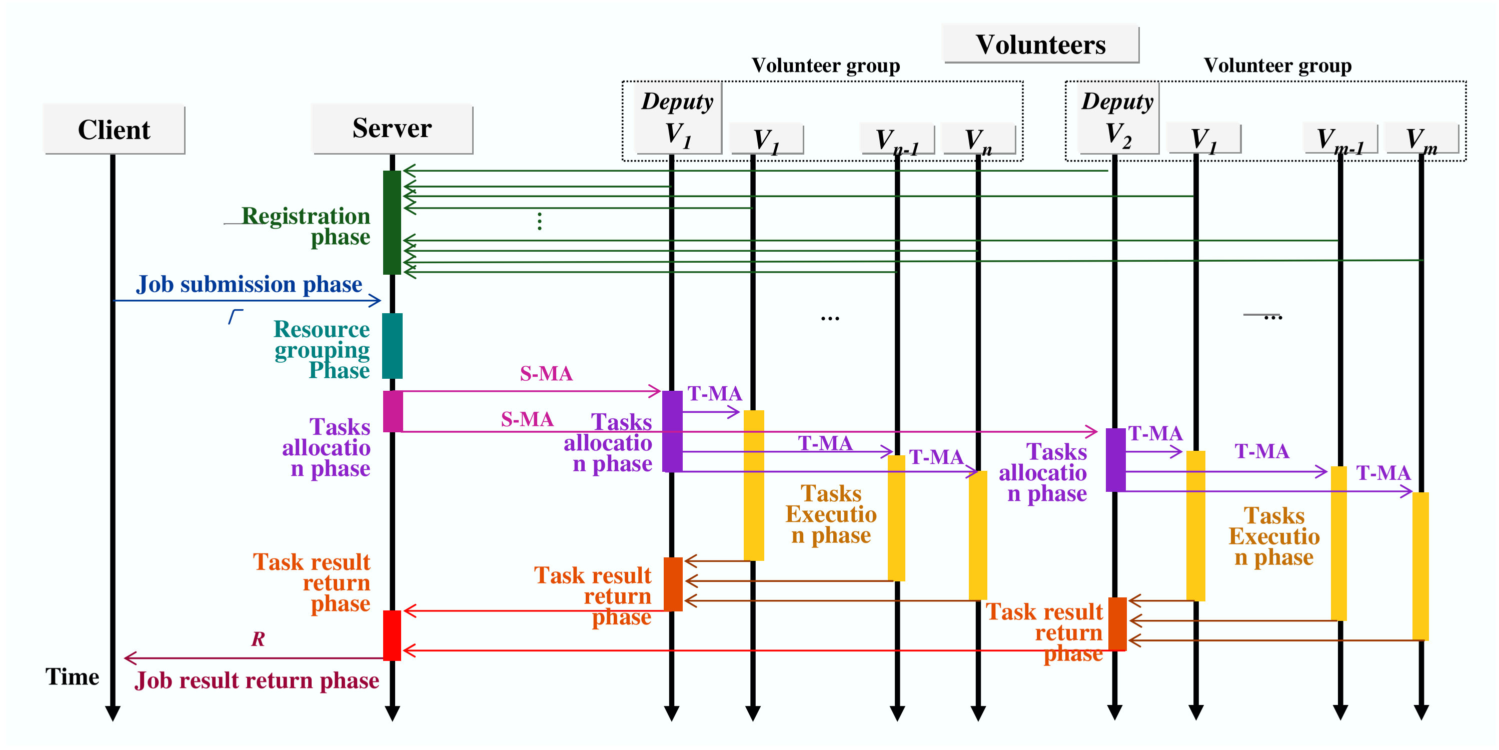


Figure 4.6: Agent based Desktop Grid computing model

In the job submission phase, the submitted job is divided into a number of tasks. The tasks are implemented as mobile agents (which are called task mobile agents).

In the resource grouping phase, a server forms volunteer groups according to properties such as location, volunteering time, availability, and credibility.

In the task allocation phase, a server does not perform entire scheduling mechanism any more. Instead, it helps mobile agents to perform scheduling procedure. Scheduling and fault tolerance algorithms are implemented as scheduling mobile agents. A server distributes scheduling mobile agents to deputy volunteers according to the properties of volunteer groups. The scheduling mobile agent distributes task mobile agents to the members of its volunteer group.

In the task execution phase, the task mobile agents are executed in cooperation with the scheduling mobile agent while migrating or being replicated to other volunteers in presence of failures.

In the task result return phase, the task mobile agent returns each result to its scheduling mobile agent. When all task mobile agents return their results, the scheduling mobile agent aggregates the results and then returns the collected results to the server.

In the job result return phase, the server returns a final result to the client when it receives all the results from the scheduling mobile agents.

4.2.3 Allocating Scheduling Agents to Scheduling Groups

After constructing volunteer groups (that is, A', B', C' and D'), a server allocates the scheduling mobile agents (S-MA) to volunteer groups. However, it is not practical to allocate S-MAs directly to the volunteer groups in a scheduling procedure because some volunteer groups are not perfect for finishing the tasks reliably. Therefore, it is necessary to build new scheduling groups by combining the volunteer groups with each other (see Table 4.1).

In Table 4.1, the first two combinations are more appropriate than the last one in the sense that the tasks are distributed to each scheduling group in the first two combinations, whereas the tasks are mainly distributed to the $A'C'$ scheduling group in the last combination. In addition, in the last combination, even though the tasks are allocated to the $B'D'$ scheduling group, they are not completed due to insufficient time. When comparing the first two combinations, the first combination is more appropriate than the second because the B' volunteer group is able to compensate for the C' volunteer group with regard to availability in the first combination, whereas the C' volunteer group does not compensate for the D' volunteer group in the second combination.[1] Therefore, this thesis focuses on the first combination in a scheduling

[1]In the $A'D'$ or the $A'B'$ scheduling groups, the A' volunteer group compensates for the D' and B' volunteer groups, because the A' volunteer group has high availability and enough Θ.

Table 4.1: The combination of volunteer groups

Combination	The number of allocated tasks	α_v compensation	Θ compensation	Description
$A'D'$ & $C'B$'	$A'D' \simeq C'B'$ or $A'D' \geq C'B'$	○	○	The tasks are distributed to each scheduling group. A' compensates for D', and C' compensates for B'.
$A'B'$ & $C'D'$	$A'B' \simeq C'D'$ or $A'B' \geq C'D'$	×	○	The tasks are distributed to each scheduling group. Both C' and D' have low α_v, so they do not compensate α_v.
$A'C'$ & $B'D'$	$A'C' \gg B'D'$	○	×	Tasks are mainly distributed to $A'C'$. Most tasks are completed in $A'C'$. Both B' and D' do not compensate Θ.

```
// $DVS$ : deputy volunteer set
// $CDVS$ : candidate deputy volunteer set
// $TDVS$ : temporal deputy volunteer set
// $CDVS \subset VG_{A'}$
$TDVS$ = OrderedBy($CDVS.\alpha_v$);
// HC : harddisk capacity, NB : network bandwidth
$DVS$ = OrderedBy($TDVS.(\Theta + CPU + DC + NB)$);
// Pop the best deputy volunteer from $DVS$
$DV$ = PopBestDV($DVS$)
```

Figure 4.7: Algorithm of deputy volunteer selection

procedure.

The S-MA is executed at a deputy volunteer. The deputy volunteer is selected using the algorithm (see Figure 4.7). The deputy volunteers are ordered by volunteer availability and volunteering service time, and also by CPU, hard disk capacity (DC), and network bandwidth (NB). Then, the deputy volunteers for scheduling groups are selected sequentially. Next, each S-MA is transmitted to the selected deputy volunteer.

4.2.4 Distributing Task Agents to Group Members

After the S-MAs are allocated to the scheduling groups, each S-MA distributes the task mobile agents (T-MA) that consist of parallel code and data to the members of the scheduling group. The S-MAs perform different scheduling, fault tolerance, and replication algorithms according to the type of volunteer groups, differently from existing Desktop

Grid computing systems.

The S-MA of the $A'D'$ scheduling group performs the scheduling as follows. 1) *Order the A' volunteer group by α_v and then by Θ.* 2) *Distribute T-MAs to the arranged members of the A' volunteer group.* 3) *If a T-MA fails, replicate the failed task to a new volunteer selected in the A' volunteer group by using the replication algorithm, or move the task to the volunteer selected in the A' or B' volunteer groups if task migration is allowed.*

The S-MA of the $C'B'$ scheduling group performs the scheduling as follows. 1) *Order the C' and B' volunteer groups by α_v and then by Θ.* 2) *Distribute T-MAs to the arranged members of the C' volunteer group.* 3) *If a T-MA fails, replicate the failed task to a new volunteer selected in the ordered C' volunteer groups, or move the task to a volunteer selected in the B' or C' volunteer groups.*

Tasks are firstly distributed to the $A'D'$ scheduling group and then the $C'B'$ scheduling group. In addition, the tasks are firstly distributed to the volunteers that have high α_v and long Θ. In the scheduling algorithm, if checkpointing is not used, tasks are not allocated to the B' and D' volunteer groups, because they have insufficient time to finish the task reliably. In this case, the B' and D' volunteer groups execute tasks for testing, that is, to measure their properties. For example, the tasks executed in the A' and C' volunteer groups are redistributed to the D' and B' volunteer groups, respectively. However, the B' volunteer group can be used to assist the main volunteer groups (that is, A' or C') if task migration is permitted. For example, in the $C'B'$ scheduling group, the B' volunteer group can be used to compensate for the C' volunteer group with regard to volunteer availability. Suppose that a volunteer in the C' volunteer group suffers from volunteer autonomy failures. If the volunteering time of a volunteer in the B' volunteer group covers the range of the duration of volunteer autonomy failures at the failed volunteer, the suspended task can migrate to the new volunteer in the B' volunteer group.

If replication is used, a S-MA calculates the number of redundancy and then selects replicas (that is, volunteers to execute the replicated computation). Then, the S-MA distributes the T-MAs to the selected replicas. In the case of failures, the S-MA replicates or moves the failed T-MA to a new volunteer. The replication and fault tolerance algorithms are described in detail, later.

4.3 Group Scheduling for Replication

Replication is a well-known technique to improve reliability and performance in distributed systems [99, 100]. In a Desktop Grid computing environment, replication is mainly used for reliability, (that is, to tolerate failures), or for result certification, (that is, to detect and tolerate erroneous results) [14, 15, 22, 55, 56, 93, 94, 95, 96, 97]. This section focuses on replication to reliably tolerate volunteer autonomy failures. Our adaptive scheduling mechanism automatically adjusts the number of redundancy, and selects an appropriate replica according to the properties of each volunteer group.

4.3.1 How to Calculate the Number of Redundancy

Our group-based adaptive replication algorithm calculates the number of redundancy on the basis of each volunteer group. In addition, it exploits volunteer autonomy failures and volunteer credibility simultaneously when calculating the number of redundancy.

In a Desktop Grid computing environment, volunteer autonomy failures occur much more frequently than crash and link failures. Therefore, we must consider volunteer autonomy failures when calculating the number of redundancy. However, existing methods [2] do not consider volunteer autonomy failures. To reflects the volunteer autonomy failures, our replication algorithm makes use of *volunteer availability* and *volunteer autonomy failures* as follows.

The number of redundancy r for reliability is calculated by Equation 4.1. Here, τ represents the $MTTVAF$ of a volunteer, and τ' represents the $MTTVAF$ of a volunteer group.

$$(1 - e^{-\Delta/\tau'})^r \leq 1 - \gamma \tag{4.1}$$

[2]Ranganathan et al. [95] proposed how to calculate the number of redundancy per files in a large peer-to-peer communities. Li and Mascagni [93] proposed the number of redundancy in computational grid.

The parameter γ is the reliability threshold.

$$\tau' = (V_1.\tau + V_2.\tau + \cdots + V_n.\tau)/n$$

Here, n is the total number of volunteers within a volunteer group. The $V_n.\tau$ means τ of a volunteer V_n.

In Equation 4.1, the expression $e^{-\frac{\Delta}{\tau'}}$ [3] represents the reliability of each volunteer group, which means the probability to complete tasks within Δ. It reflects volunteer autonomy failures. The $(1 - e^{-\frac{\Delta}{\tau'}})^r$ means the probability that all replicas fail to complete the replicated tasks.

Each volunteer group has different r. For example, the A' and C' volunteer groups have smaller r than the B' volunteer group. Similarly, the A'' and C'' volunteer groups have smaller r than the B'' volunteer group.

In a Desktop Grid computing environment, some malicious volunteers tamper with the computation and then return corrupted results. In this case, the Desktop Grid computing systems must detect and tolerate the erroneous result, which is called *result certification.* To this end, majority voting and spot-checking have been exploited [4]. In this sec-

[3]If the lifetime of a volunteer is exponentially distributed, then the reliability of the volunteer $R(t)$ is : $R(t) = e^{-\lambda' t}$ [99, 100, 101]. The parameter λ' refers to the rate of volunteer autonomy failures. If the probability that tasks are completed at time interval Δ is calculated, then the $e^{-\frac{\Delta}{\tau'}}$ is also calculated because $\frac{1}{\lambda'} = \tau'$.

[4]Sarmenta [23, 24] proposed how to calculate the number of redundancy for majority voting. However, he does not consider volunteer credibility and volunteer

tion, we focus on majority voting. Especially, we calculate the number of redundancy for majority voting by using *volunteer credibility*.

The number of redundancy r for majority voting is dynamically calculated through Equation 4.2 [102]. Here, $r = 2k + 1$ [5].

$$\sum_{i=k+1}^{2k+1} \binom{2k+1}{i} (1 - C'_v)^i (C'_v)^{(2k+1-i)} \leq 1 - \vartheta \tag{4.2}$$

The parameter ϑ is the desired credibility threshold that a task achieves. The parameter C'_v means the probability that volunteer group generates correct results.

$$C'_v = (V_1.C_v + V_2.C_v + \cdots + V_n.C_v)/n$$

In Equation 4.2, the left expression $\sum_{i=k+1}^{2k+1} \binom{2k+1}{i} (1-C'_v)^i (C'_v)^{(2k+1-i)}$ represents the error probability to each volunteer group, which is bounded by the following equation [102].

$$\frac{[4C'_v(1 - C'_v)]^{k+1}}{2(2C'_v - 1)\sqrt{\pi k}}$$

Each volunteer group has the different number of redundancy. For example, the A' and B' volunteer groups have smaller r than C' volunteer group. Similarly, the A'' and B'' volunteer groups have smaller r than C'' volunteer group.

autonomy failures. In addition, he does not provide replication mechanism on a per group basis

[5]If k malicious volunteers return the erroneous results (that is, volunteers exhibit Byzantine failures), a minimum of $2k + 1$ volunteers are needed to achieve k fault tolerance [99, 100].

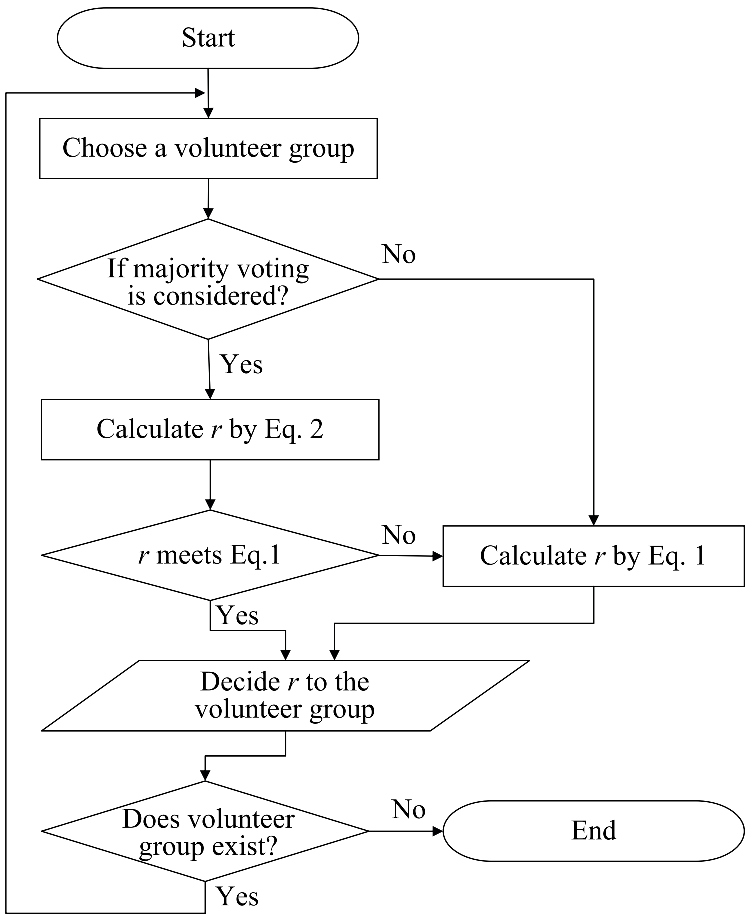


Figure 4.8: Algorithm for calculating the number of redundancy

To apply volunteer autonomy failures and volunteer credibility simultaneously when calculating the number of redundancy, we propose the following algorithm as shown in Figure 4.8.

4.3.2 How to Select Replicas

After deciding the number of redundancy to each volunteer group, our group-based adaptive replication algorithm selects replicas (that is, volunteers to execute the replicated task) according to the number of redundancy. Therefore, each volunteer group has many *replication groups*, which refer to a set of replicas for a task.

Selection in Classification I.

To make a replication group for a task, volunteers within each volunteer group are sorted by volunteer availability α_v and volunteering service time Θ. Especially, A' and C' volunteer group is sorted by α_v and then by Θ. B' volunteer group is sorted by Θ and then by α_v. The Θ is important because of insufficient volunteering service time in B' volunteer group. After each volunteer group is sorted, the replication groups are constructed according to r.

Selection in Classification II.

To make a replication group for a task, volunteers within each volunteer group are sorted by volunteer availability α_v, volunteering service time Θ, and volunteer credibility C_v. Especially, A'' volunteer group is sorted by α_v and then by Θ. The C_v does not matter because the value is beyond the desired credibility ϑ in A'' volunteer group. B'' volunteer group is sorted by Θ and then by α_v. The Θ is important because of insufficient volunteering service time in B'' volunteer group. C'' volunteer group is sorted by C_v and the by α_v because it has low credibility. After each volunteer group is sorted, the replication groups are constructed according to r.

Fault Tolerant Selection in Classification I & II.

When a volunteer suffers from failures, the failed volunteer is replaced by a new one. In our replication mechanism, the failed volunteers are replaced by volunteers with higher or equal quality in order to keep the credibility and availability higher or equal. In the classification I, the failed volunteers in C' volunteer group are replaced by new volunteers in A' or C' volunteer group. In the classification II, the failed volunteers in C'' volunteer group are replaced by new volunteers in A'' or C'' volunteer group.

4.3.3 How to Distribute Tasks to Replicas

The method to distribute a task to replication group is categorized into two approaches: parallel distribution and sequential distribution as shown in Figure 4.9.

In Figure 4.9, the replication group consists of volunteers, V_0, V_1, and V_2 (that is, $r = 3$). With the parallel distribution, the task (T_i) is distributed to all members at the same time as shown in Figure 4.9 (a), and then executed simultaneously. On the other hand, with the sequential distribution, the task (T_i) is distributed and executed sequentially as shown in Figure 4.9 (b).

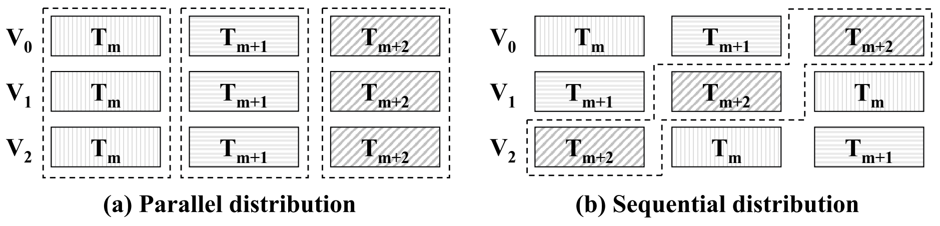


Figure 4.9: Parallel and sequential distribution

Distribution in Classification I.

In the case of A' volunteer group, sequential distribution is more appropriate than parallel one because the former can complete more tasks. For example, in Figure 4.9 (b), if V_0 completes the task T_m, there is no need to execute it at V_1 and V_2. A' volunteer group has high possibility to execute a task reliably without failures (especially, volunteer autonomy failures) because of high volunteer availability. However, if A' volunteer group performs parallel distribution as shown in Figure 4.9 (a), it exhibits overhead of replication in the sense that the volunteers execute only the same tasks even though it is able to execute other tasks. In contrast to A' volunteer group, in case of C' volunteer group, sequential distribution is more appropriate than parallel one because C' volunteer group frequently suffers from volunteer autonomy failures owing to low α_v.

Distribution in Classification II.

In the case of A'' volunteer group, sequential distribution is more appropriate than parallel one because the former can perform more tasks. That is, A'' volunteer group has high possibility to produce correct results, so it can carry out its task reliably without failures (especially, volunteer autonomy failures). For example, if V_0 completes the task T_i and its reliability is satisfied in Figure 4.9 (b), there is no need to execute it at V_2. In addition, in the case of majority voting, if the first two results of T_{i+2} generated at V_1 and V_2 are the same, there is no need to execute the T_{i+2} at V_0 as shown by the dotted line in Figure 4.9 (b) because majority (that is, 2 out of 3) is already achieved. Therefore, the volunteers can execute other tasks as soon as majority is reached, instead of the tasks indicated by the solid line in Figure 4.9 (b).

In the case of B'' volunteer group, sequential distribution is more appropriate than parallel one, just like with the A'' volunteer group. However, B'' volunteer group has low Θ, so it can not complete their tasks because of the lack of the computation time. Therefore, the manager of B'' volunteer group should provide task migration in order to execute the tasks continuously. During task migration, a former volunteer affects the new volunteer to which a task is migrated. In other words, if the malicious volunteer is wrongly selected as the new volunteer, it ruins the correct result that was generated by the former volunteer. Therefore, the new volunteer must be chosen among B'' or

A'' volunteer group, not C'' or D'' volunteer group.

In the case of C'' volunteer group, sequential distribution is more appropriate than parallel one. C'' volunteer group has enough time to execute tasks. However, it has low credibility, so it has a high probability that its result is incorrect. Moreover, each volunteer suffers from volunteer autonomy failures owing to low α_v. In the case of majority voting, the voting procedures are delayed if sequential distribution is adopted. In other words, it takes a longer time and high overhead to complete the result certification. In the case of parallel distribution, the overhead and time are smaller than in sequential distribution relatively because voting procedure for each task is completed within a step as shown in Figure 4.9 (a).

4.4 Group Scheduling for Result Certification

Result certification approaches are categorized into majority voting and spot-checking mechanisms [15, 23, 24, 56, 98]. Our group-based scheduling mechanism dynamically applies result certification to volunteer groups.

4.4.1 Applying Result Certification to Volunteer Group

Result certification is dynamically applied to each volunteer group.

The A'' volunteer group has high C_v, high Θ, and high α_v enough to execute tasks reliably. In the case of majority voting, sequential distribution is more appropriate than parallel one, as in section 4.3.3. The A'' volunteer group performs spot-checking smaller than C'' volunteer group.

The B'' volunteer group has high C_v and high α_v, but low Θ. In the case of majority voting, sequential voting group is more appropriate than parallel voting group as with A'' volunteer group. If migration occurs, spot-checking is additionally performed at a former volunteer as well as migrated volunteer to check their correctness.

The C'' volunteer group has high Θ, but low C_v and low α_v. Thus, its results might be incorrect. The C'' volunteer group should do more spot-checking in order to strength the credibility. Parallel voting group is more appropriate than sequential voting group, as in section 4.3.3.

4.4.2 Scheduling Algorithm for Result Certification

The tasks are scheduled in the following order, that is, A'', C'' and B'' volunteer groups sequentially, because A'' and C'' volunteer groups have enough times to execute tasks.

In the A'' volunteer group, scheduling algorithm for result certification is as follows: 1) Order A'' volunteer group by α_v and then by Θ. 2) Evaluate the number of redundancy or spot-checking rate. 3) Construct a sequential voting group, or choose some volunteers for spot-checking on the basis of Θ. 4) Distribute tasks in a way of sequential voting group, or allocate special tasks for spot-checking. 5) Check the collected results.

In the B'' volunteer group, scheduling algorithm for result certification is as follows: 1) Order A'' volunteer group by Θ and then by α_v. 2) Same to A'' volunteer group. 3) Construct a sequential voting group, or choose some volunteers for spot-checking on the basis of Θ. 4)$\sim$5) are same as A'' volunteer group. Especially, B'' volunteer group must perform additional spot-checking during task migration because of lack of volunteering service time.

In the C'' volunteer group, scheduling algorithm for result certification is as follows: 1) Order C'' volunteer group by C_v and then α_v. 2) Evaluate the number of redundancy or spot-checking rate. 3) Construct a parallel voting group, or choose some volunteers for spot-checking on the basis of C_v. 4)$\sim$5) are the same as A'' volunteer group. C'' volunteer group should handle the volunteer autonomy failures.

In the above 2) phases, the number of redundancy for majority voting and the number of spot-checking are differently applied to each volunteer group. In case of redundancy for majority voting, C'' volunteer group has the greater number of redundancy than A'' and B'' volunteer groups because of low credibility. Similarly, C'' volunteer group has the greater number of spot-checking than A'' and B'' volunteer groups. Especially, B'' volunteer group has the more number of spot-checking than A'' volunteer group on account of task migration.

The number of redundancy r for majority voting is dynamically calculated through the following Equation 4.3. Here, $r = 2k + 1$. The final error rate of majority voting is evaluated as follows [24, 102].

$$\varepsilon(C'_v, r) = \sum_{i=k+1}^{2k+1} \binom{2k+1}{i} (1 - C'_v)^i (C'_v)^{(2k+1-i)} \tag{4.3}$$

which is bounded by $\frac{[4C'_v(1-C'_v)]^{k+1}}{2(2C'_v-1)\sqrt{\pi k}}$.

Here, the parameter C'_v means the probability that each volunteer group generates correct results. The C'_{vA},C'_{vB}, and C'_{vC} is the C'_v of A'', B'', and C'', respectively.

Supposed that a desired credibility threshold is ϑ, our group-based adaptive result certification algorithm calculates the the redundancy of each volunteer group r_A, r_B and r_C (that is, the redundancy of A'', B'', and C'', respectively) as follows.

If $(1 - \vartheta) \geq \varepsilon(C'_{vA}, r_A)$, we decide r_A as the number of redundancy of A'' volunteer group. The r_B and r_C are also calculated in the same way. So, r_A and r_B is smaller than r_C. Consequently, A'' or B'' volunteer group has the small number of redundancy, so it can reduce the overhead of majority voting and execute more tasks. In contrast, C'' volunteer group has the large number of redundancy. The large redundancy not only makes the credibility high, but also compensates for low availability (that is, tolerates volunteer autonomy failures). Finally, the desired error rate is satisfied entirely because $(1 - \vartheta) \geq \varepsilon(C'_{vA}, r_A)$.

The rate of spot-checking q is also calculated. The final error rate of spot-checking is evaluated as follows [24].

$$\varepsilon(q, n, C'_v, s) = \frac{sC'_v(1-qs)^n}{(1-C'_v) + C'_v(1-qs)^n} \tag{4.4}$$

where, n is the saboteur's share in the total work. s is the sabotage rate of a saboteur.

Our group-based adaptive result certification algorithm calculates the spot-checking rate of each volunteer group q_A, q_B and q_C (that is, A'', B'', and C'', respectively) as follows.

In a similar way of majority voting, if n and s are given, the spot-checking rate q_A of A'' volunteer group is calculated by Equation 4.4. If $(1-\vartheta) \geq \varepsilon(q_A, n, C'_{vA}, s)$, we decide q_A as the spot-checking rate of A'' volunteer group. Since C'_{vA} and C'_{vB} are greater than C'_{vC}, the spot-checking rates of A'' and B'' group, q_A and q_B, are smaller than the the spot-checking rate of C'' group, q_C. Therefore, A'' and B'' volunteer groups can reduce the overhead and execute more tasks. The C'' volunteer group can increase its credibility.

4.5 Fault Tolerant Algorithm

Volunteer autonomy failures lead to the delay and blocking of the execution of tasks. They occur much more frequently than crash and link failures in a Desktop Grid computing environment. Moreover, volunteers take various occurrence rates and forms of volunteer autonomy failures. A Desktop Grid system is required to conduct various fault tolerance algorithms in scheduling procedures according to the occurrence rate and form. To achieve this, we apply different fault tolerance algorithms according to the property of each volunteer group, while also distinguishing volunteer autonomy failures from the traditional failures. We describe how the scheduling and task mobile agents work in the presence of failures in this subsection.

The volunteer autonomy failures Φ are different from crash failure in that the operating system is alive in spite of volunteer volatility failure Φ and volunteer interference failure Ψ, whereas it shuts down in the presence of crash failure [58, 99, 100]. Φ is different from crash failure in that Φ occurs due to the request of volunteers [58, 99, 100]. Ψ is different from Φ in that a Desktop Grid computing system is alive in spite of Ψ, whereas it is not operating in the case of Φ.

A server detects the crash failure of S-MA using a timeout. Similarly, the S-MA detects the crash failure of T-MA. To achieve this, the S-MA sends alive messages to its server. Similarly, the T-MA sends alive messages to the S-MA. The T-MAs in the D' volunteer group[6] do not send alive messages, in order to reduce the management overhead. A volunteer can detect volunteer autonomy failures by oneself because its operating system does not shut down. If T-MA or S-MA detects the volunteer autonomy failures, it notifies its S-MA or server, respectively.

[6] D'' volunteer group in the Classification II.

4.5.1 Handling Failure of Scheduling Agent

A S-MA rarely suffers from volunteer autonomy failures because it is executed at the deputy volunteers that are selected among the A' volunteer group[7]. The S-MA stores information such as scheduling group lists, scheduling table, and task results in a stable storage. If the S-MA fails, the information is sent to a new deputy volunteer. Figure 4.10 shows the fault tolerant algorithm of S-MA.

If a server detects the crash failure of S-MA, the new deputy volunteer is selected by the algorithm of deputy volunteer selection presented in Figure 4.10. Next, the S-MA and the scheduling information are sent to the newly selected deputy volunteer.

If a S-MA suffers from the volunteer volatility failure, it sends a *VolatilityFailure* message to the server. If the S-MA joins again during the volunteering time, it sends *Rejoin* message to its server. If the server does not receive a *Rejoin* message within the interval after receiving a *VolatilityFailure* message, it sends the S-MA to a new deputy volunteer.

If a S-MA is at the edge of reserved volunteering time, it sends an *InAdvanceVolatilityFailure* message to its server. In this case, the server responds with a candidate deputy volunteer. The S-MA migrates to the candidate deputy volunteer.

In the case of volunteer interference failure, a S-MA does not take any action because it can perform scheduling procedures in the sense

[7] A'' volunteer group in the Classification II.

```
[ If a server detects the crash failure of S-MA ]
$V_m$ = SelectDeputyVolunteer($A'$);
SendS-MA$'$($V_m$);  // send S-MA$'$(the latest checkpointed S-MA) to $V_m$

[ If $\Phi$ occurs ]
// At the S-MA side
S-MA$'$ = Checkpoint(S-MA);
Send VolatilityFailure message to server;
Send S-MA$'$ to server;
// At the server side
if (Server is informed of the volunteer volatility failure) then
    if (Server does not received the Rejoin message within the interval) then
        $V_m$ = SelectDeputyVolunteer($A'$);
        SendS-MA$'$($V_m$);
    fi;
fi;

[ At the edge of volunteering time ]
// At the S-MA side
S-MA$'$ = Checkpoint(S-MA);
Send InAdvanceVolatilityFailure message to server;
if (S-MA receivs the candidate deputy volunteer $V_m$) then
    MigrateS-MA$'$($V_m$);
fi;
// At the Server side
if (S-MA receives InAdvanceVolatilityFailure message) then
    $V_m$ = SelectDeputyVolunteer($A'$);
    SendCandidateDeputyVolunteer($V_m$);
fi;
```

Figure 4.10: Fault tolerant algorithm in the presence of failures of S-MA

that the Desktop Grid system is alive.

4.5.2 Handling Failure of Task Agent

A T-MA suffers from volunteer autonomy failures more frequently than a S-MA, because the volunteer running a T-MA has relatively low availability. The T-MA checkpoints the execution state at the rate of $MTTVAF$ if checkpointing is used. Figures 4.11, 4.12, and 4.13 show the fault tolerant algorithm of T-MA[8].

If a S-MA detects the crash failure of T-MA, it selects a new volunteer. If checkpointing is used, the S-MA sends the latest checkpointed T-MA$'$ to it. Otherwise, the S-MA redistributes the T-MA to the new one. Each S-MA redistributes the T-MA within the number of redundancy r.

If a T-MA is at the edge of reserved volunteering time, it sends a *InAdvanceVolatilityFailure* message to its S-MA. After receiving a candidate volunteer, it migrates to the candidate volunteer or is replicated.

If a T-MA suffers from volunteer volatility failure Φ, it takes a checkpoint of the execution of task and then notifies its S-MA of Φ by means of a *VolatilityFailure* message. Next, if the S-MA does not receive any *Rejoin* message from the failed volunteers within predefined time interval, it reschedules the T-MA. If checkpointing and migration are used, the S-MA migrates the T-MA$'$ to a new volunteer. Otherwise, the S-MA replicates the T-MA by the number of redundancy r.

[8]In the figures, A', B', and C' are replaced by A'', B'', and C'', respectively, when considering the Classification II.

```
 [ If S-MA detects the crash failure of T-MA ]
$V_m$ = SelectNewVolunteer();
if (checkpointing is used) then
    SendT-MA$'$($V_m$);   //  send T-MA$'$(the latest checkpointed T-MA) to $V_m$
else
    RedistributeT-MA($V_m$);   //  redistribute T-MA to $V_m$
fi;

[ At the edge of reserved volunteering time ]
// At the T-MA side
if (the task is not finished) then
    if  (T-MA $\in A'$ || T-MA $\in C'$ || T-MA $\in B'$) then
        Send *InAdvanceVolatilityFailure* message to S-MA;
    fi;
    if (T-MA receives the candidate volunteer $V_m$) then
        if (checkpointing is used) then
            MigrateT-MA$'$($V_m$);   //  migrate T-MA$'$ to $V_m$
        else
            ReplicateT-MA($V_m$);   //  replicate T-MA to $V_m$
        fi;
    fi;
fi;
// At the S-MA side
if  (S-MA receives *InAdvanceVolatilityFailure* message) then
    $V_m$ = SelectNewVolunteer();
    SendCandidateVolunteer($V_m$);
fi;

[ Function : SelectNewVolunteer() ]
if  (T-MA $\in A'$ volunteer group) then
    $V_m$ = SelectNewVolunteer($A'$);
else if  (T-MA $\in C'$ volunteer group) then
    $V_m$ = SelectNewVolunteer($C'$, $B'$);
else if  (T-MA $\in B'$ volunteer group) then
    $V_m$ = SelectNewVolunteer($B'$);
    SendT-MA($V_m$);
fi;
```

Figure 4.11: Fault tolerant algorithm in the presence of failures of T-MA (1)

```
 [ If $\Phi$ occurs ]
// At the T-MA side
T-MA$'$ = Checkpoint(T-MA);
if (T-MA $\in A'$ || T-MA $\in C'$ || T-MA $\in B'$) then
    Send VolatilityFailure message to S-MA;
    Send T-MA$'$ to S-MA;
fi;
// At the S-MA side
if (S-MA receives VolatilityFailure message) then
    if (S-MA does not receive the Rejoin message within the interval)
then
        $V_m$ = SelectNewVolunteer();
        if (checkpointing is used) then
            SendT-MA$'$($V_m$);
        else
            ReplicateT-MA($V_m$);
        fi;
    fi;
fi;
```

Figure 4.12: Fault tolerant algorithm in the presence of failures of T-MA (2)

If a T-MA suffers from volunteer interference failure Ψ, it takes a checkpoint of the execution. Then, if the execution is not restarted within the interval, the volunteer sends an *InterferenceFailure* message to its S-MA. After receiving a candidate volunteer, the T-MA migrates to the candidate volunteer or is replicated.

In the algorithm, there is no fault tolerant mechanism for the D'

```
[ If $\Psi$ occurs ]
// At the T-MA side
T-MA$'$ = Checkpoint(T-MA);
if (T-MA is not restarted within the interval) then
    if (T-MA $\in A'$ || T-MA $\in C'$ || T-MA $\in B'$) then
        Send InterferenceFailure message to S-MA;
    fi;
    if (T-MA receives the candidate volunteer $V_m$) then
        if (checkpointing is used) then
            SendT-MA$'$($V_m$);
        else
            ReplicateT-MA($V_m$);
        fi;
    fi;
fi;
// At the S-MA side
if (S-MA receives InterferenceFailure message) then
    $V_m$ = SelectNewVolunteer();
    SendCandidateVolunteer($V_m$);
fi;
```

Figure 4.13: Fault tolerant algorithm in the presence of failures of T-MA (3)

volunteer group[9] in the presence of failures during the execution in order to reduce management overhead. The D' volunteer group[10] executes the task for testing, for example, for the purpose of recalculating volunteer

[9] D'' volunteer group in the Classification II.

[10] D'' volunteer group in the Classification II.

autonomy failures, volunteer availability, and volunteering service time.

4.6 Related Work

4.6.1 Scheduling and Fault tolerance

AgentTeamwork [83] proposed a mobile agent based PC Grid middleware. AgentTeamwork makes use of mobile agents for resource search and job migration in the presence of crash failure. AgentTeamwork is similar to our system in the sense that it uses mobile agents. However, our mechanism uses mobile agents for scheduling. The scheduling mobile agent is permitted to implement different scheduling algorithms suitable for each volunteer group according to volunteering service time and volunteer availability.

Kondo et al. [47, 49] classified clients (that is, volunteers in this thesis) into two classes mainly according to location (that is, home or workplace) and network: *conservative* and *extreme*. Conservative clients model home PCs. They have relatively slow network bandwidth and are used sparsely during after-work hours. Extreme clients, however, model PCs in the workplace. They have relatively fast broadband network and are used all day. In addition, conservative clients are 90% idle, whereas extreme clients are 80% idle. Kondo et al. [47, 49] measured scheduling mechanisms (i.e., timeout and duplication) for five client groups with various conservative/extreme percentages. Even though classified clients were used, the classification was used to set up simulation environments. However, our adaptive scheduling mechanism directly exploits classification to apply various scheduling, fault tolerance, and replication strategies to each volunteer group by means of mobile agents. In addition, our classification and scheduling mechanism focuses on volunteer autonomy failures, volunteering service time, and volunteer availability at the same time in a scheduling procedure, whereas the strategies employed by Kondo et al. were mainly based on CPU capacity, location, and network state in a scheduling procedure.

In the SETI@Home [12] project, the central server has the role of scheduling and management of volunteers, and therefore has high overhead. A volunteer takes a checkpoint periodically (every ten minutes). Each time volunteers are interrupted by the user or a system failure, the computation will resume from the last saved checkpointing. The SETI@Home project is based on the BOINC [13] that is middleware for public resource computing. The BOINC provides redundant computing to deal with erroneous computational results [14, 15]. However, SETI@Home and BOINC do not provide a scheduling mechanism on a per group basis.

The XtremWeb [17, 18, 19] proposed a FIFO scheduling scheme. Bayanihan [22, 23, 24] proposed a sabotage-tolerance mechanism to tolerate malicious volunteers. It tolerates erroneous results from malicious volunteers by using majority voting and spot-checking mechanisms. The majority voting and spot-checking are based on eager scheduling algorithm. Javelin [25, 26, 27] proposed an advanced eager scheduling algorithm based on tree. However, XtremWeb, Bayanihan, and Javelin do not propose scheduling on the basis of volunteer groups. They also do not exploit volunteer autonomy failures, volunteering service time, and volunteer availability in a scheduling phase. In addition, they reduce the overhead of central server by using the multiple servers, whereas our adaptive scheduling mechanism makes use of mobile agents.

The CCOF [34, 35, 36] proposed a wave scheduler in which nodes (that is, volunteers) are classified into night zone and day zone. The wave scheduler is based on timezone-aware overlay network using CAN. In the wave scheduler, when morning arrives at a node, it randomly selects a node in a nightzone for the purpose of migration. However, CCOF does not consider volunteer groups that are constructed by volunteer autonomy failures and volunteer availability during classification. In addition, it does not apply different scheduling, fault tolerance, and replication strategies to each timezone.

Maheswaran et al. [67] proposed dynamic matching and scheduling heuristics for independent tasks in heterogeneous computing systems (that is, computational grid). They evaluated on-line mode heuristics (MCT, MET, SA, and KPB) and batch mode mapping heuristics (Min-min, Max-min, and sufferage). Casanova et al. [84] modified existing scheduling heuristics (Min-min, Max-min, and Sufferage) in a grid computing environment. Input and output data transfer time is considered when computing MCT (Minimum Completion Time), and locality of files is also taken into account. They also proposed XSufferage in which the sufferage value is computed with cluster-lever MCT. Berman et al. [85] proposed application level scheduling (AppLeS) for adaptive application scheduling in a grid computing environment. The AppLeS uses agents that are customized for particular applications on potential target resources by means of the performance model. However, existing mechanisms are based on the computational grid computing environment. They do not provide scheduling, fault tolerance, and replication algorithms on the basis of volunteer groups. They mainly focus on CPU load and network bandwidth in a scheduling procedure, whereas our adaptive scheduling mechanism concentrates on volunteer availability, volunteering time, and volunteer autonomy failures that occur more frequently in a Desktop Grid computing environment.

In Condor [44, 45, 46], checkpointing is used for a task to migrate from a non-idle workstation to an idle workstation (in this case, the two workstations should be homogeneous). Condor proposed ClassAds for matchmaking and a priority based scheduling algorithm. However, Condor is better suited for intra-organizational use, whereas a Desktop Grid computing system is based on the Internet. In addition, Condor does not apply different scheduling, fault tolerance, and replication strategies to each volunteer group.

Kondo et al. [48] proposed two kinds of availability: host availability and CPU availability. Host availability refers to the probability indicating whether a host is reachable and up. CPU availability means the probability that quantifies the fraction of the CPU that can be used by an application. In addition, they estimated CPU availability experimentally. Bhagwan et al. [92] empirically characterized host availability in a P2P computing environment. Our volunteer availability is similar to CPU availability. However, we conceptually classify and define volunteer availability that is derived from volunteer autonomy failures. In addition, we provide a volunteer group based scheduling mechanism to directly exploit the availability.

4.6.2 Scheduling for Replication

Replication is a well-known technique to improve reliability and performance in distributed systems [99, 100]. Some studies have been made on replication in a grid computing environment or a P2P computing environment [23, 49, 55, 93, 94, 95, 96, 97].

Li and Mascagni [93] proposed computational replication to improve performance in large-scale computational grid. They propose how to determine the number of task replicas to meet the performance goals on the basis of node and network failure rates. Kondo et al. [47, 49] proposed computational replication (i.e., duplication and timeout mechanisms) in a desktop grid computing environment. With the duplication mechanism, each task is replicated by the maximum number of redundancy. With the timeout mechanism, each task is replicated if its result is not returned within the predefined timeout. Ranganathan and Foster [94] proposed dynamic replication strategies in a data grid environment. They provide six different strategies to replicate large amounts of data for the purpose of reducing bandwidth consumption and access latency.

Ranganathan et al. [95] proposed dynamic model-driven replication to improve data availability in a large peer-to-peer communities. They provide the methods not only to compute the number of replicas per file, but also to determine the location for a new replica on the basis of storage and transfer costs. Cohen and Shenker [96] proposed replication strategies in unstructured peer-to-peer networks. They proposed uniform, proportional, and square-root replication to minimize the expected search size. Cuenca-Acuna et al. [97] proposed replication to increase the availability of shared data in unstructured peer-to-peer systems. They replicate files using an erasure code.

Sarmenta [23, 24] proposed sabotage-tolerance mechanism for volunteer computing systems. The proposed mechanism tolerates erroneous results from malicious volunteers by using majority voting and spot-checking mechanisms. Especially, with majority voting, the same task is replicated at different volunteers as much as the number of redundancy to meet the desired error rate.

Most replication approaches focus on data replication (that is, replicating data or file), whereas Li and Mascagni [93], Kondo et al. [47, 49], and Sarmenta [23, 24] deal with computational replication (that is, replicating the execution of task). Data replication is mainly used to improve data availability and access time in a peer-to-peer network or a data grid computing environment. On the other hand, computational replication is mainly used for fault tolerance or result certification in a computational grid or a desktop grid computing environment. In particular, Li and Mascagni [93] and Kondo et al. [47, 49] used replication for fault tolerance and performance, whereas Sarmenta [23, 24] used replication for result certification (that is, tolerating malicious volunteers). In addition, existing replication mechanisms [23, 24, 47, 49, 93] are not on a per group basis.

4.6.3 Scheduling for Result Certification

In a Desktop Grid computing environment, the result certification approaches are categorized into majority voting and spot-checking mechanisms [15, 23, 24, 56, 98].

In a majority voting approach, the same task is carried out at different volunteers as much as the number of redundancy. Redundancy is used to identify the correct result against erroneous one if there are sufficiently more good volunteers than bad ones. In a spot-checking approach, the special task whose result is already known is performed at randomly selected volunteers. If a volunteer returns an erroneous result, it is regarded as malicious one. Majority voting approach is apparently more costly than spot-checking, because it requires a redundancy of at least two per task.

In addition, the Desktop Grid computing systems mainly used the eager scheduling algorithm [22, 25, 27, 29]. In this algorithm, the tasks are continuously allocated to faster volunteers, so they execute more tasks than slow volunteers.

A few studies have been made on scheduling and result certification in a Desktop Grid computing environment. Bayanihan [22, 23, 24] proposed majority voting or spot-checking based on eager scheduling algorithm. Especially, it also proposed credibility-enhanced eager scheduling algorithm. In this algorithm, the more a volunteer passes the spot-checking, the higher its credibility becomes. The more volunteers within voting group agree on a result, the higher its credibility becomes. Volunteers continue to compute the task and perform spot-checking until the credibility threshold is satisfied. When the desired credibility threshold is reached, the result is accepted as a final. In these algorithms, the voting group for majority voting is not built before tasks are distributed to volunteers. Instead, the voting group is built on the fly. That is, whenever a faster volunteer is allocated to a task, it is added to the voting group for the task. As a result, the members in the voting group have different credibility. Renaud and Playez [99] proposed a spot-checking mechanism on the basis of property testing. They attempted to reduce the sample size for spot checking by using property testing. Taufer et al. [15] provides homogeneous redundancy for the numerical applications that may produce different results depending on the machine architecture, operating systems, compiler, and compiler flags.

Chapter 5

Performance Evaluation

We evaluate the performance of our group-based adaptive scheduling mechanism through simulation. We implemented our group-based adaptive scheduling mechanism and configured volunteer's properties by using data collected from Korea@Home [52]. In our simulation, 200 volunteers have different volunteering service time, volunteer availability, and volunteer credibility. They are classified into groups, that is, (A', B', C', D') or (A'', B'', C'', D'') by the group construction algorithms as shown in Figure 4.3, 4.5 in Chapter 4. We measure our group-based adaptive scheduling, replication, and result certification algorithms in terms of the number of the competed tasks, the number of redundancy, spot-checking rate, and error rate.

5.1 Group-based Adaptive Scheduling

We evaluate the group-based adaptive scheduling mechanism with existing scheduling mechanisms. The evaluation focuses on how much performance improvement is achieved, depending on whether volunteer groups are considered in a scheduling procedure. To this end, volunteer groups were intentionally set up, which have different volunteering service time Θ and volunteer availability α_v. We compare our adaptive scheduling mechanism with eager scheduling. In eager scheduling[1] [16, 18, 22, 25, 30], a volunteer asks its volunteer server of a new task as soon as it finishes its current task. As a result, the more eagerly a volunteer works, the more tasks are executed. There are a lot of scheduling heuristics in grid computing environments, for examples, MCT, MET, SA, KPB, min-min, max-min, and sufferage heuristics [67, 68, 84]. In this thesis, we adopt eager scheduling among existing scheduling heuristics because it is more straightforward and simple than other heuristics in grid computing. In particular, the eager scheduling has been used mainly in dynamic Desktop Grid computing environments [16, 18, 22, 25, 30] because it is more adaptive to dynamic environments than other heuristics in grid computing.

We make use of a simulation to evaluate the proposed scheduling mechanism. The simulation was conducted on the basis of Korea@Home [52]. A task in the application consumes 16 minutes of execution time on a dedicated Pentium 1.4 GHz. Table 5.1 presents the simulation environment with different volunteer groups, volunteering service time, and volunteer availability. For each case in Table 5.1, 200 volunteers

[1]Eager scheduling is very similar to FCFS(First Come First Served) or FIFO (Fist In First Out).

participated in the simulation during one hour. In Case 1, the A' volunteer group has more volunteers than the other groups. Case 2 shows that more volunteers belong to the A' and C' volunteer groups when compared to the other groups. In Case 3, the A' and B' volunteer groups have more volunteers than the other groups. In Case 4, the D' volunteer group has more volunteers than the other groups. When analyzing Table 5.1, it can be observed that Case 1 has larger volunteer availability and volunteering service time than the other cases. Case 4 has smaller volunteer availability and volunteering service time than the other cases. Based on this simulation environment, the simulation is conducted 10 times per each case.

As shown in Table 5.1, the 200 volunteers have various volunteer autonomy failures, volunteer availability, and volunteering service time. We assume that the range of $MTTVAF$ is $1/0.2 \sim 1/0.02$ minutes and $MTTR$ is $3 \sim 10$ minutes. The simulation uses the number of completed tasks and the number of redundancy as the performance metrics. In addition, we measure the number of completed tasks depending on whether replication is applied or not. We measure two performance metrics on the basis of scheduling groups (that is, $A'D'$ and $C'B'$).

Figure 5.1 presents the average number of completed tasks. In Figure 5.1, ES and AS represent existing eager scheduling and our group-based adaptive scheduling, respectively. In addition, AS($A'D'$) and AS($C'B'$) represent each scheduling group in our group-based adaptive scheduling

Table 5.1: Simulation environment

Case		A'	B'	C'	D'	Total
Case1	# of vol.	127 (63%)	30 (15%)	35 (17%)	9 (5%)	200
	α_v	0.95	0.95	0.74	0.77	0.91
	Θ	43	15	31	11	35 min.
Case2	# of vol	95 (47%)	26 (13%)	63 (32%)	16 (8%)	200
	α_v	0.9	0.9	0.65	0.65	0.80
	Θ	40	14	28	9	30 min.
Case3	# of vol	78 (39%)	75 (37%)	16 (8%)	31 (16%)	200
	α_v	0.95	0.95	0.70	0.61	0.88
	Θ	31	11	25	8	20 min.
Case4	# of vol	52 (26%)	48 (24%)	23 (12%)	77 (38%)	200
	α_v	0.85	0.85	0.56	0.54	0.70
	Θ	28	9	22	7	15 min.
# of vol.: the number of volunteers						

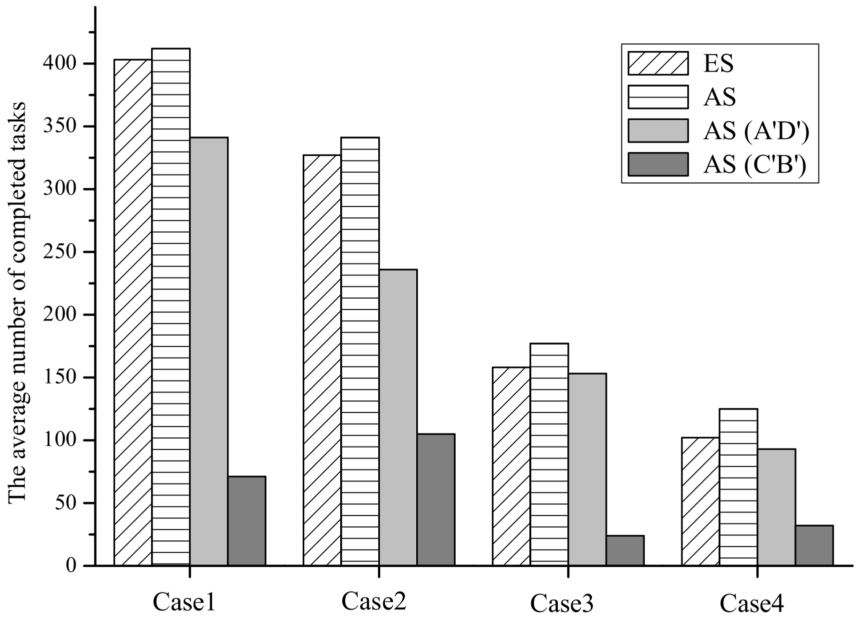


Figure 5.1: The average number of completed tasks

(Note that the sum of AS($A'D'$) and AS($C'B'$) is equal to AS). As presented in Figure 5.1, our group-based adaptive scheduling completes more tasks than the existing eager scheduling.

The obtained results indicate the following factors. First, the A' volunteer group has an important role in gaining better performance. When the number of members in the A' volunteer group decreases gradually (that is, from Case 1 to Case 4), the number of completed tasks also decreases.

Second, the number of members of the A' and C' volunteer groups is more important than that of the B' and D' volunteer groups. For example, Cases 1 and 2 have more completed tasks than Cases 3 and 4.

Third, volunteer availability is tightly related with performance improvement. For instance, Case 1 with the highest volunteer availability has completed many tasks than the other cases. On the other hand, the completed tasks of Case 4 with the lowest volunteer availability are less than those of the other cases.

Finally, as the number of members in the A' volunteer group gradually decreases and the number of members in the B' and D' volunteer groups increases, the difference between our group-based adaptive scheduling and the eager scheduling increases. This result is anticipated in the sense that, in the eager scheduling, the failed or suspended tasks in A', B', C', or D' volunteer groups can be redistributed to low quality volunteers interchangeably. On the other hand, since our group-based adaptive scheduling performs scheduling on a per group basis, the undesired situation does not happen. For example, the failed or suspended tasks in the C' volunteer groups are not distributed to the B' and D' volunteer groups. The difference in Case 1 is smaller than other cases because there are more members of the A' volunteer group than other groups. In other words, the undesired situations rarely occur in Case 1.

Figure 5.2 presents the average number of completed tasks when replication is used to tolerate volunteer autonomy failures for Case 2. In Figure 5.2, the tick value 1.0 on the x-axis actually represents 0.99

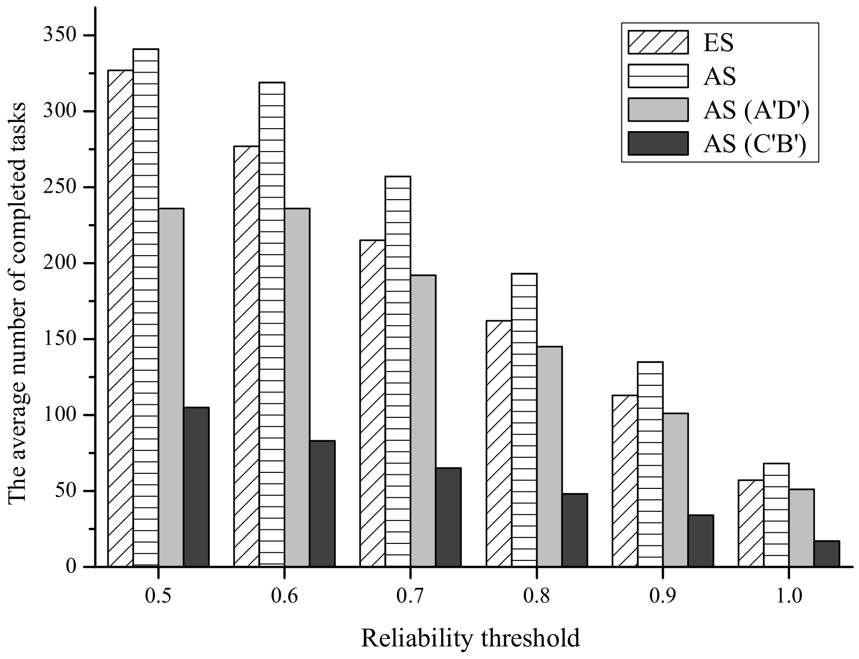


Figure 5.2: The average number of completed tasks in the case of replication in Case 2

(refer to Equation 4.1 in Chapter 4). As shown in the Figure 5.2, as the reliability threshold increases, the number of completed tasks decreases. The obtained results indicate that more tasks should be replicated to support higher reliability.

Figure 5.3 presents the number of redundancy r for Case 2. our group-based adaptive scheduling has a smaller r than the eager scheduling because the scheduling mobile agent applies the replication algorithm to each volunteer group. That is, it adaptively adjusts the number

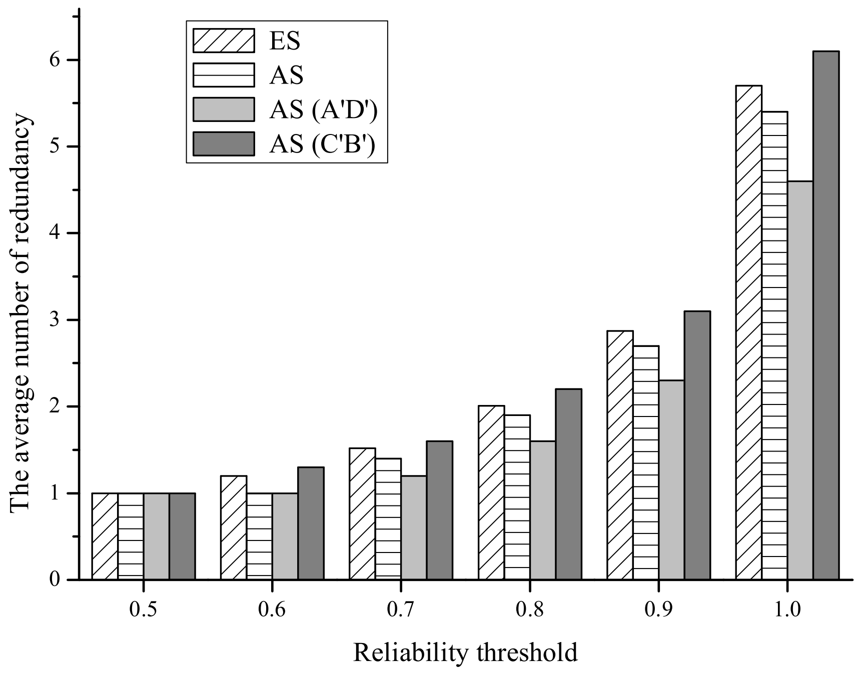


Figure 5.3: The average number of redundancy in Case 2

of redundancy r according to the rate of volunteer autonomy failures of volunteer groups. In addition, the $A'D'$ scheduling group has a smaller r than the $C'B'$ scheduling group because the A' volunteer group has higher volunteer availability and volunteering service time than the C' volunteer group. Since the C' volunteer group suffers from volunteer autonomy failures more frequently than the A' volunteer group, the former has a greater r than the latter. Therefore, in the case of the A' volunteer group, the small r satisfies the reliability threshold. In the case of the C' volunteer group, the large r is required to meet the reliability threshold. As a result, the A' volunteer group can execute more tasks because it can reduce replication overhead. Finally, as the reliability is increasingly required, the number of redundancy r increases.

Figure 5.4 presents the average number of completed tasks in the case of replication. In Figure 5.4, the value of 0.8 is used as the reliability threshold. When compared to Figure 5.1, the difference between our group-based adaptive scheduling and the eager scheduling is larger. In our group-based adaptive scheduling, the A' volunteer group can complete more tasks, because it has a relatively small r. On the other hand, the eager scheduling does not consider a homogeneous group, so the following undesirable situation occurs repeatedly. Suppose that a volunteer in the C' volunteer group suffers from volunteer autonomy failures. In this case, its failed task should be distributed to a new volunteer. In the eager scheduling, the new volunteer is selected without considering volunteer groups. If the newly selected volunteer belongs to the B' or D' volunteer groups, it would also fail because of the high rate of volunteer autonomy failures. If volunteers with low quality are selected continuously, the task is continuously redistributed to other volunteers until a high quality volunteer is chosen. Such an undesirable situation occurs frequently and repeatedly if there are a lot of volunteers belonging to the B', C', or D' volunteer groups. Thus, the difference between our group-based adaptive scheduling and the eager scheduling in the Cases 3 and 4 is larger than that in Cases 1 and 2.

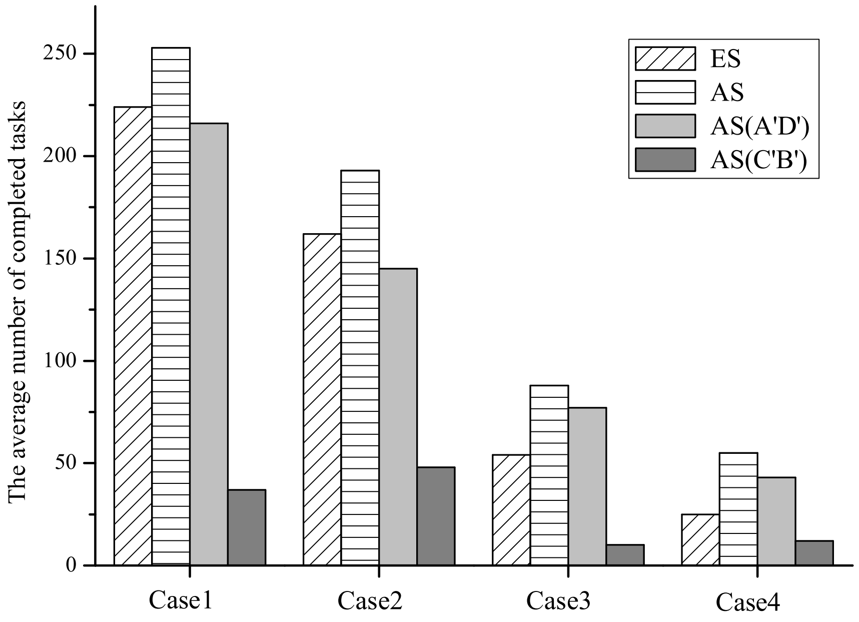
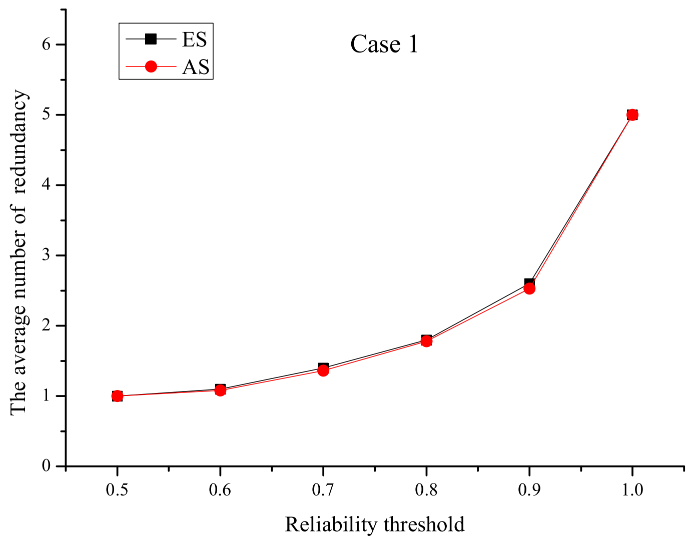


Figure 5.4: The average number of completed tasks in case of replication (reliability threshold = 0.8)

Figure 5.5 and 5.6 present the number of redundancy r for all cases. As the number of members in A' volunteer group decreases, the difference between our group-based adaptive scheduling and the eager scheduling increases. For example, Case 1 has the largest A' volunteer group, therefore, the number of redundancy r of our group-based adaptive scheduling is similar to that of eager scheduling. Since Case 2 has many members of the A' and C' volunteer groups, the gap between our group-based adaptive scheduling and the eager scheduling is larger than that shown in Case 1. Similar results are presented in Cases 3



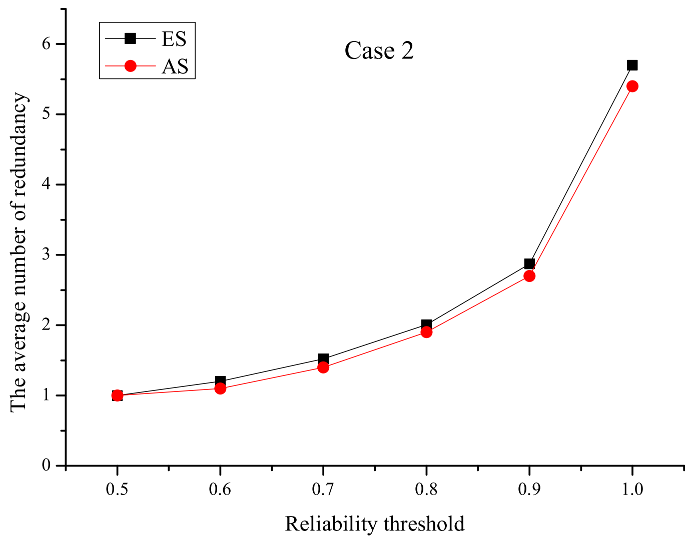
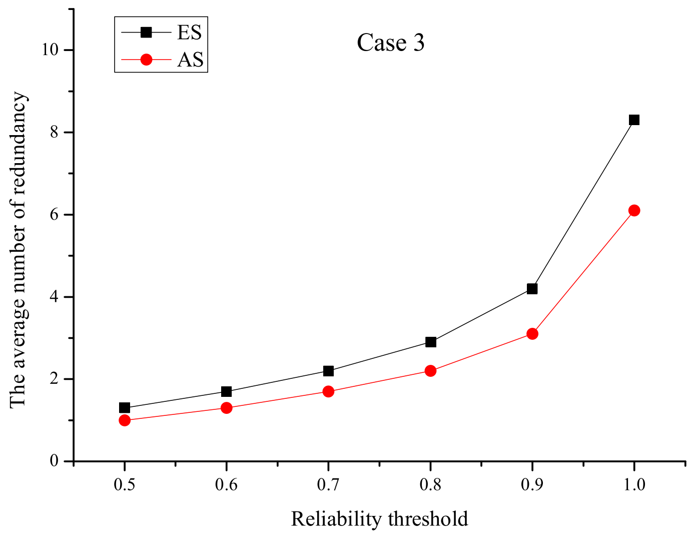


Figure 5.5: The average number of redundancy



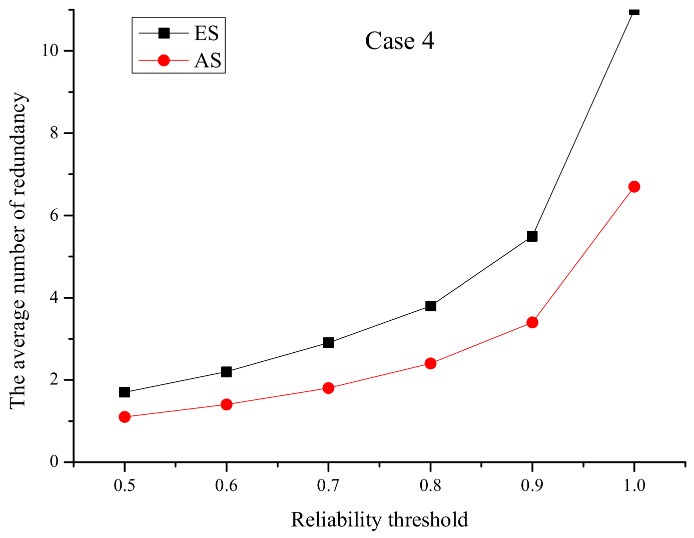


Figure 5.6: The average number of redundancy (Continued)

and 4. Compared with the eager scheduling, our group-based adaptive scheduling has a small r because it calculates the number of redundancy on the basis of volunteer groups, in contrast to eager scheduling. In our group-based adaptive scheduling, volunteer groups with a high rate of volunteer autonomy failures require a large r, and vice versa. Consequently, our group-based adaptive scheduling completes more tasks than the eager scheduling. A' volunteer group can complete more tasks because it has a smaller number of redundancy than the eager scheduling as presented in Figure 5.5 and 5.6.

5.2 Group Scheduling for Replication

We evaluate our group-based scheduling for replication mechanism with existing replication mechanisms [23, 47, 93]. The evaluation focuses on how much performance improvement is achieved, depending on whether volunteer groups (A'', B'', C'', D'') are considered. To achieve this, volunteer groups are set up, which have different volunteering service time, volunteer availability, volunteer credibility as described in Table 5.2.

The 200 volunteers have various volunteer availability, volunteer credibility, and volunteering service time as shown in Table 5.2. Here, $P.$ (that is, population) represents the number of volunteers. We assume that the range of $MTTVAF$ is $1/0.2 \sim 1/0.02$ minutes, and $MTTR$ is $3 \sim 10$ minutes. A task consumes 16 minutes of execution time on a

Table 5.2: Simulation environment for replication

Case		A''	B''	C''	D''	Total
Case1	P.	103(51.5%)	18 (9%)	64 (32%)	15 (7.5%)	200
	α_v	0.86	0.87	0.79	0.86	0.84
	Θ	42	17	39	17	37 min.
	C_v	0.98	0.98	0.87	0.89	0.94
Case2	P.	72 (36%)	24 (12%)	74 (37%)	30 (15%)	200
	α_v	0.84	0.80	0.78	0.75	0.80
	Θ	36	16	32	16	29 min.
	C_v	0.98	0.98	0.87	0.80	0.91
Case3	P.	33 (16.5%)	47 (23.5%)	60 (30%)	60 (30%)	200
	α_v	0.83	0.81	0.78	0.74	0.79
	Θ	35	13	36	13	23 min.
	C_v	0.98	0.98	0.87	0.86	0.91

dedicated Pentium 1.4 GHz. In Table 1, the volunteering service time Θ is decreasing from Case 1 to Case 3. Case 1 is smaller than Case 2 with respect to volunteer availability and volunteer credibility. Case 2 is different from Case 3 with respect to volunteering service time. Cases 1 and 2 have more A'' and C'' volunteer groups than B'' and D'' volunteer groups. On the other hand, Case 3 have more D'' volunteer groups than A'' and B'' volunteer groups. Each simulation was repeated 10 times per case.

Figure 5.7 shows the average number of redundancy r when the result certification (that is, majority voting) is considered. The r is calculated by the algorithm as shown in Figure 4.8. We assume that the credibility threshold ϑ is 0.98 and γ is 0.5. In this case, the r is affected by volunteer credibility C_v. First of all, our group-based replication mechanism has lower r than the mechanisms that calculates r without volunteer group. Second, when comparing Case 1 with Case 2 and Case 3, the r increases because Case 1 has greater than Case 2 and 3 with respect to C_v. Finally, the difference between two bars in Case 3 is greater than that in Case 2 not only because the population of A'' and C'' volunteer groups in Case 3 is smaller than that in Case 2, but also because the Θ in Case 3 is smaller than that in Case 2.

Figure 5.8 shows the number of completed tasks when result certification is considered. The ϑ and γ are the same as Figure 5.7. Our adaptive replication mechanism calculates r to each volunteer group, and then distributes tasks to volunteer groups by using distribution ap-

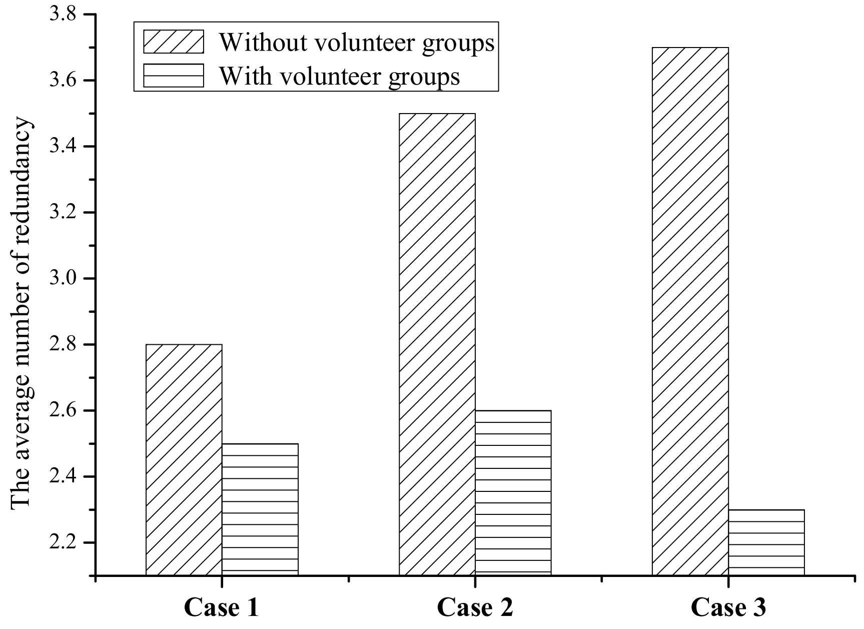


Figure 5.7: The average number of redundancy with majority voting

proaches (that is, parallel distribution and sequential distribution). As a result, it completes more tasks than the existing mechanisms that do not consider volunteer groups and distribution approaches. Case 1 has higher α_v, Θ and C_v than Case 2 and Case 3, so it has the larger number of completed tasks than the latter.

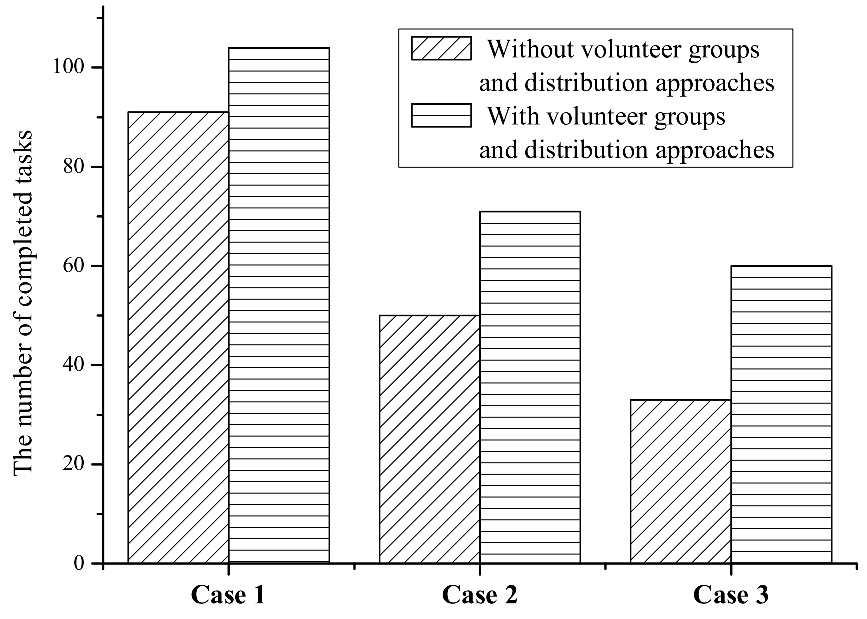


Figure 5.8: The number of completed tasks with majority voting

5.3 Group Scheduling for Result Certification

We compare our group-based scheduling mechanism with eager scheduling mechanism for evaluating result certification. This simulation focuses on how much we obtain performance improvement according to whether the volunteer groups are considered in a scheduling procedure on the basis of Θ, α_v and C_v, or not.

We evaluated the 200 volunteers with respect to 4 cases during 1 hour as shown in Table 5.3. Case 1 is different from Case 2 with respect to

Table 5.3: Simulation environment for result certification

Case		A''	B''	C''	D''	Total
Case1	P.	84 (42%)	26 (13%)	70 (35%)	20 (10%)	200
	α_v	0.84	0.88	0.81	0.83	0.84
	Θ	41	17	39	16	35 min.
	C_v	0.98	0.98	0.88	0.86	0.93
Case2	P.	71 (35.5%)	31 (15.5%)	76 (38%)	22 (11%)	200
	α_v	0.87	0.89	0.80	0.82	0.84
	Θ	41	17	39	16	34 min.
	C_v	0.98	0.98	0.84	0.85	0.91
Case3	P.	76 (38%)	27 (13.5%)	80 (40%)	17 (8.5%)	200
	α_v	0.86	0.78	0.80	0.71	0.81
	Θ	35	17	33	16	30 min.
	C_v	0.98	0.98	0.82	0.85	0.91
Case4	P.	42 (21%)	59 (29.5%)	30 (15%)	69 (34.5%)	200
	α_v	0.80	0.70	0.78	0.69	0.73
	Θ	28	12	25	13	24 min.
	C_v	0.98	0.98	0.89	0.89	0.94

the volunteer credibility. Case 3 is different from Case 1 with regard to volunteer availability and volunteer availability. Case 4 is different form Case 1 with respect to volunteer availability and volunteering service time. Here, A'', B'', C'', D'' represent A'', B'', C'', D'' volunteer groups, respectively. Each simulation experiment was repeated 10 times per each Case.

The 200 volunteers have various volunteer availability, volunteer credibility and volunteering service time as shown in Table 5.3. We assume that the range of MVT is 10 $\sim$ 60 min., $MTTVAF$ = 1/0.2 $\sim$ 1/0.05 min., $MTTR$ is 3 $\sim$ 10 min. We assume that the expected execution time of a task at volunteers is 18 $\sim$ 20 minutes. Suppose that s=0.1 and n=10 in spot-checking.

Figures 5.9, 5.10, 5.11, and 5.12 show the simulation results. Here, ES represents existing eager scheduling mechanism. AS represents our adaptive scheduling mechanism. AS(A''), AS(B'') and AS(C'') mean the results performed in each volunteer group, respectively. As shown in Figures 5.9 and 5.10, our adaptive scheduling mechanism for result certification completes more tasks than existing eager scheduling mechanism, while satisfying the desired credibility threshold (or desired error rate) as shown in Figure 5.11(b) and 5.12(b).

In the case of majority voting, our scheduling mechanism obtains more results of tasks than eager scheduling because it dynamically decides the number of redundancy according to properties of volunteer groups as shown in Figure 5.11(a). A'' and B'' volunteer groups choose

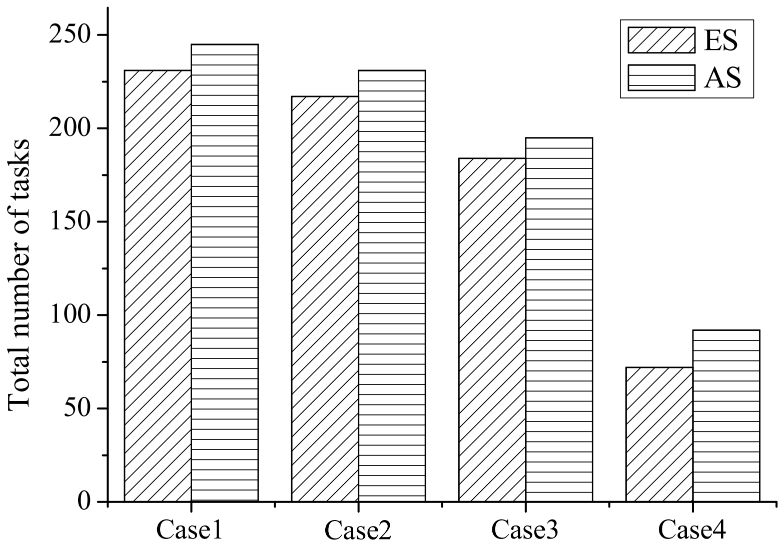
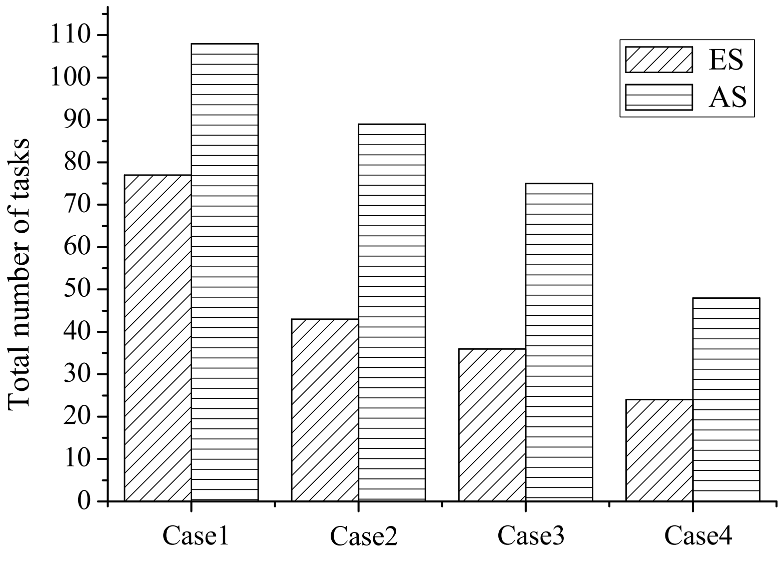


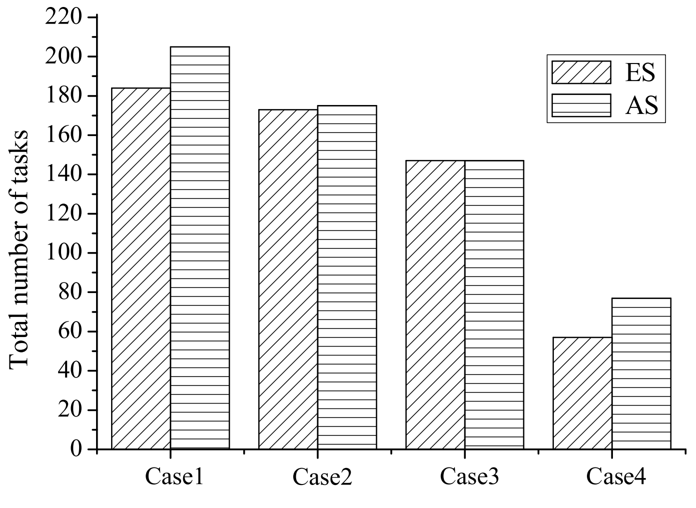
Figure 5.9: Total number of task without result certification

less redundancy than C'' volunteer group. As a result, A'' and B'' volunteer groups are able to reduce the overhead, so they work more. On the other hand, C'' volunteer group can decrease its error rate.

In the case of spot-checking, our scheduling mechanism completes more tasks than eager scheduling because it dynamically decides spot-checking rate according to properties of volunteer groups as shown in Figure 5.12(a). However, if A'' volunteer group is less than C'' volunteer group like Case 2 and 3, the the number of tasks becomes similar because spot-checking does not much generates overhead in comparison with majority voting.

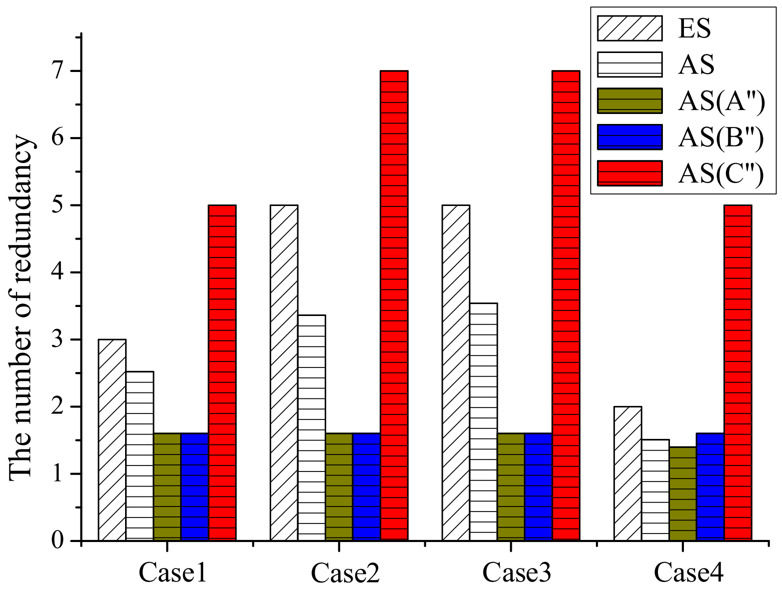


(a) Total number of tasks with majority voting

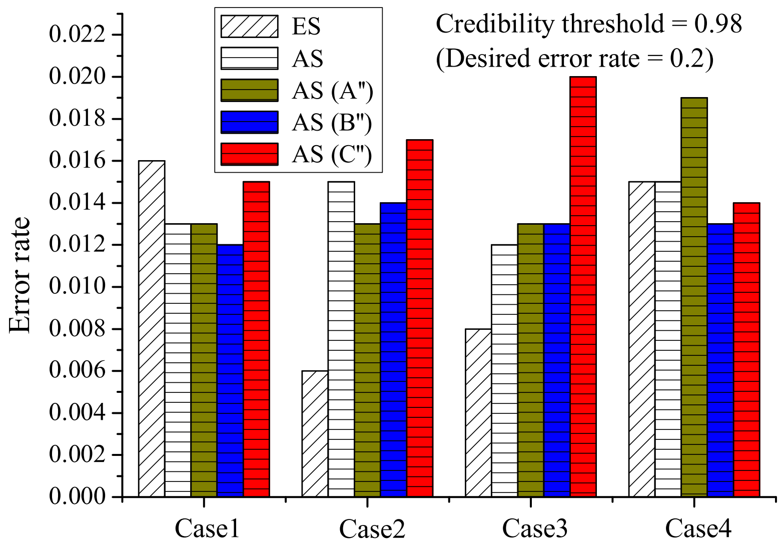


(b) Total number of tasks with spot-checking

Figure 5.10: Total number of task with result certification

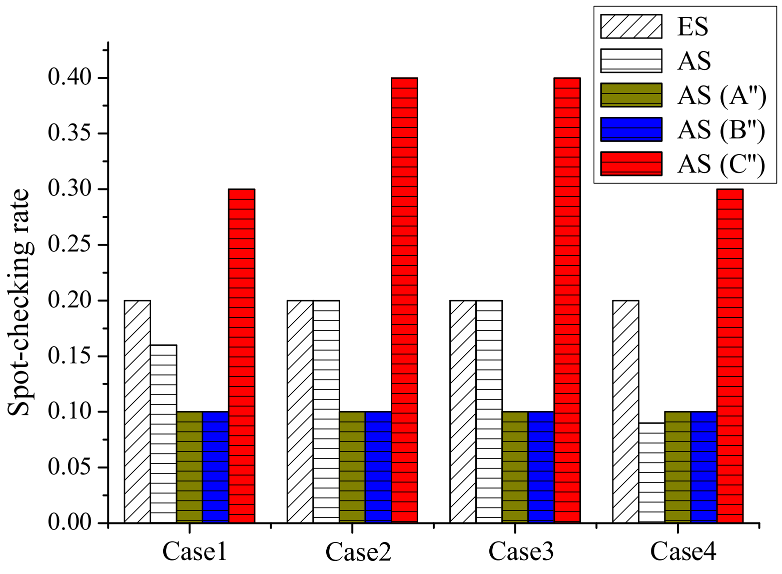


(a) The number of redundancy in majority voting

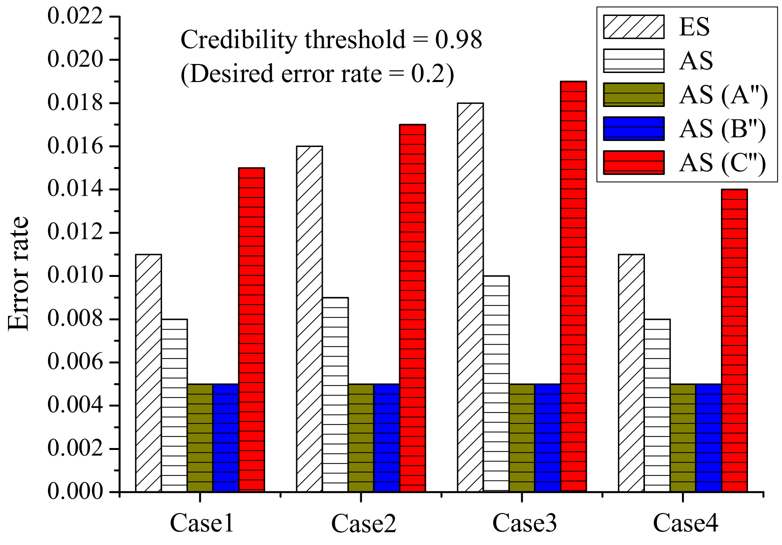


(b) Error rate in majority voting

Figure 5.11: Result certification: majority voting



(a) Spot-checking rate



(b) Error rate in spot-checking

Figure 5.12: Result certification: spot-checking

Chapter 6

Conclusion

In this chapter, we summarize the contributions of this thesis and discuss various avenues of future work.

6.1 Contributions

This thesis focuses on characterizing and categorizing the aspects of Desktop Grid systems. To this end, we proposed the taxonomy of Desktop Grid focusing on scheduling. We also presented a mapping of the taxonomy to the Desktop Grid systems. From the taxonomy and survey, we extracted the challenging issues for Desktop Grid scheduling such as volatility, dynamic environment, lack of trust, failure, heterogeneity, scalability, and voluntary participation. To overcome these challenges, we proposed a new direction for Desktop Grid scheduling such as 1) resource grouping, 2) reputation or incentive-based scheduling, 3) scheduling for result certification, 4) dynamic, adaptive, or fault tolerant scheduling, and 5) distributed scheduling. Among these directions, this thesis focused on the resource grouping, result certification, replication, adaptive and fault tolerant scheduling.

For the purpose, we proposed a group-based adaptive scheduling mechanism in a Desktop Grid computing environment. The proposed scheduling mechanism considers the volatility, credibility, and heterogeneous properties of volunteers in a scheduling procedure in the sense that they are tightly related with the computation completion, reliability, and performance. The proposed scheduling mechanism constructs volunteer groups according to the properties of volunteers such as volunteer autonomy failures, volunteer availability, volunteering service time, and volunteer credibility. It dynamically applies different scheduling, replication, result certification, fault tolerance algorithms to each volunteer group.

The evaluation results demonstrated that the proposed scheduling mechanism obtains better performance and reduces the overhead of computation. Particularly, the proposed scheduling mechanism completes more tasks than eager scheduling. In the case of replication, the proposed scheduling mechanism completes more tasks than the eager scheduling because it dynamically adjusts the number of redundancy, and selects replicas on the basis of the properties of each volunteer group. With regard to volunteer groups, the evaluation results demonstrated that the larger the number of volunteers in the A' and C' volunteer groups is, the larger the number of completed tasks becomes. As the number of volunteers in the B' and D' volunteer groups increases, the difference between the proposed scheduling mechanism and the eager scheduling increases. We found that our group-based scheduling mechanism for result certification completes more tasks than existing eager scheduling mechanism, while satisfying the desired credibility threshold. Particularly, A'' and B'' volunteer groups choose less redundancy and spot-checking rate than C'' volunteer group. Consequently, A'' and B'' volunteer groups are able to reduce the overhead, so they work more.

To sum up, this thesis has the following contributions:

1. This thesis discusses the key concepts and characteristics about Desktop Grid. It also provides a new comprehensive taxonomy and survey of Desktop Grid.

2. This thesis presents the key functionalities that Desktop Grid scheduling should support. It provides comprehensive taxonomy and survey of Desktop Grid scheduling.

3. This thesis proposes a resource grouping method, which classifies and constructs groups according to volunteer's properties such as dedication, volatility, availability, and credibility.

4. This thesis proposes a new group-based adaptive scheduling mechanism, which adapts to a dynamic Desktop Grid computing environment. The proposed scheduling mechanism couples resource grouping with scheduling. It applies various replication, result certification, and fault tolerant algorithms to each homogeneous

group.

6.2 Future Work

Among the directions for Desktop Grid scheduling, this thesis focused on resource grouping, result certification, replication, adaptive and fault tolerant scheduling. Now, we are working on the rest of the directions: reputation or incentive-based scheduling and distributed scheduling.

In Desktop Grid, resources can be eager, volatile, selfish, or malicious. In order to score and rank the resources, and then reward or punish them according to the assessment, reputation/incentive-based scheduling is necessary. Reputation-based scheduling can choose more high qualified resources, so that it can improve the reliability and performance. An incentive-based scheduling focuses on punishing (for example, exclusion) volatile, selfish, or malicious resources. We are developing a new reputation/incentive-based scheduling, which inspires volunteers to donate their resources reliably and eagerly.

Distributed Desktop Grid has recently emerged as an alternative of centralized Desktop Grid because it can solve the overhead and scalability of centralized Desktop Grid. However, distributed Desktop Grid needs to construct computational overlay network(CON) for efficient scheduling, and to equip with a distributed scheduling because there is no central server. Existing decentralized Desktop Grid systems provide how to construct a CON depending on timezone, performance, or workload, but they do not consider volatility, volunteering time, credibility, and reputation/trust, which directly affect reliability, completion time, and result correctness. They also do not consider these properties during scheduling. Particularly, they do not provide replication or result certification mechanism. We will study and develop a new CON construction method and a new distributed scheduling algorithm, which are coupled with replication and result certification.

Bibliography

[1] F. Berman, G. C. Fox, and A. J. G. Hey, *Grid Computing : Making the Global Infrastructure a Reality*, Chapter 1,2,3,6,8, Wiley, 2003.

[2] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new Computing Infrastructure*, Chapter 1,2,3,4,11,17,18,28,29, Morgan Kaufmann, 2004.

[3] M. Li and M. Baker, *The Grid: Core Technologies*, Chapter 1,2,6,7, John Wiley & Sons Ltd, 2004.

[4] M. Baker, R. Buyya, and D. Laforenza, "Grids and Grid Technologies for wide-area distributed computing," *Software: Practice and Experience*, Vol. 32, Issue 15, pp. 1437-1466, Dec. 2002.

[5] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," *The 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Feb. 2003.

[6] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," *HP Laboratories Palo Alto HPL-2002-57*, Mar. 2002.

[7] D. Barkai, *Peer-to-Peer computing: Technologies for Sharing and Collaborating on the Net*, Intel Press, 2002.

[8] R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications*, Chapter 2-8,13,16, Springer, 2005.

[9] R. Subramanian and B. D. Goodman, *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*, Chapter 1,11, IDEA group Publishing, 2005.

[10] The Great Internet Mersenne Prime Search (GIMPS), "http://www.mersenne.org/"

[11] Distributed.net, "http://distributed.net"

[12] SETI@home, "http://setiathome.ssl.berkeley.edu"

[13] BOINC (Berkeley Open Infrastructure for Network Computing), http://boinc.berkeley.edu/

[14] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," *The Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, IEEE CS Press, pp. 4-10, Nov. 2004.

[15] M. Taufer, D. Anderson, P. Cicotti, and C. L. Brooks III, "Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing," *The 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Heterogeneous Computing Workshop (HCW'05)*, pp. 119a, Apr. 2005.

[16] D. P. Anderson, E. Korpela, and R. Walton, "High-Performance Task Distribution for Volunteer Computing," *The First IEEE International Conference on e-Science and Grid Technologies (e-Science2005)*, pp. 196-203, Dec. 2005.

[17] XtremWeb, http://www.lri.fr/ fedak/XtremWeb/

[18] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: A Generic Global Computing System," *The 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001): Workshop on Global Computing on Personal Devices*, pp. 582-587, May 2001.

[19] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri, and O. Lodygensky, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid," *Future Generation Computer Systems*, vol. 21, issue 3, pp. 417-437, Mar. 2005

[20] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, issue 5, pp. 597-610, May 2003.

[21] A. A. Chien, S. Marlin, and S. T. Elbert, "Resource management in the Entropia System," *Chapter 26 in Grid Resource Management: Sate of the Art and Future Trends*, Kluwer Academic, 2003.

[22] L. F. G. Sarmenta and S. Hirano. "Bayanihan: building and studying web-based volunteer computing systems using Java," *Future Generation Computer Systems, Special Issue on Metacomputing*, vol. 15, issue 5-6., Oct. 1999.

[23] L. F. G. Sarmenta, "Sabotage-Tolerance Mechanisms for Volunteer Computing Systems," *Future Generation Computer Systems*, vol. 18, issue 4, pp. 561-572, Mar. 2002.

[24] L. F. G. Sarmenta, "Volunteer computing," *Ph.D. Thesis*, Department of Electrical Engineering and Computer Science, MIT, Jun. 2001.

[25] M. O. Neary, B. O. Christiansen, P. Cappello, and K. Schauser, "Javelin: Parallel computing on the internet," *Future Generation Computer Systems, Special Issue on Metacomputing*, vol. 15, issue 5-6, pp. 659-674, Oct. 1999.

[26] M. O. Neary, S. P. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability Issues in Global Computing," *Concurrency: Parctice and Experience*, vol. 12, issue 8, pp. 727-753, Aug. 2000.

[27] M. O. Neary and P. Cappello, "Advanced eager scheduling for Java-based adaptive parallel computing," *Concurrency and Computation: Practice and Experience*, vol. 17, issue 7-8, pp. 797-819, Jun. 2005.

[28] R. Buyya and S. Vazhkudai, "Compute Power Market: towards a market-oriented grid," *The First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pp. 574-581, May 2001.

[29] T. T. Ping, G. C. Sodhy, C. H. Yong, F. Haron, and R. Buyya, "A Market-based Scheduler for JXTA-based Peer-to-Peer Computing System," *International Conference on Computational Science and its Applications (ICCSA 2004)*, LNCS 3046, pp.147-157, May 2004.

[30] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff, "Charlotte: Metacomputing on the Web," *Future Generation Computer Systems, Special Issue on Metacomputing*, vol. 15, issues 5-6, pp. 559-570, Oct. 1999.

[31] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over the Internet-the POPCORN project," *The 18th International Conference on Distributed Computing Systems*, pp. 592-601, May 1998.

[32] J. P. Morrison, J. J. Kennedy, and D. A. Power, "WebCom: A Web Based Volunteer Computer," *Journal of Supercomputing*, vol. 18, no. 1, pp. 47-61, Jan. 2001.

[33] J. P. Morrison, J. J. Kennedy, and D. A. Power, "Load balancing and fault tolerance in a condensed graphs based metacomputer,"

The Journal of Internet Technologies, Special Issue on Web based Programming, vol. 3, no. 4 , pp. 221-234, Dec. 2002.

[34] D. Zhou and V. Lo, "Cluster Computing on the Fly: resource discovery in a cycle sharing peer-to-peer system," *IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*, pp. 66-73, May 2004.

[35] V. Lo, D. Zhou, D. Zappala, Y. Liu, and S. Zhao, "Cluster Computingon the Fly: P2P Scheduling of Idle Cycles in the Internet," *The 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, LNCS 3279, pp.227-236, Feb. 2004.

[36] D. Zhou and V. Lo, "Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-to-peer Desktop Grid Systems," *The 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, LNCS 3834, pp. 194-218, Jun. 2005.

[37] S. Zhao and V. Lo, "Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems," *The Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pp. 31-38, Sept. 2005.

[38] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 3, pp. 1-12, May 2005.

[39] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "The Organic Grid: Self-Organizing Computational Biology on Desktop Grids," *Chapter 27 in Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, Wiley, 2006.

[40] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: a framework for the development of agent-based peer-to-peer systems," *The 22nd International Conference on Distributed Computing Systems*, pp. 15-22, Jul. 2002.

[41] A. Montresor, H. Meling, and O. Babaoglu, "Messor: Load-Balancing through a Swarm of Autonomous Agents," *International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, LNCS 2530, pp. 125-137, Jul. 2003.

[42] L. Zhong, D. Wen, Z. W. Ming, and Z. Peng, "Paradropper: a general-purpose global computing environment built on peer-to-peer overlay network," *The 23rd International Conference on Distributed Computing Systems (ICDCS 2003), Workshop on New Advances of Web Server and Proxy Technologies (NAWSPT)*, pp. 954-957, May 2003.

[43] W. Dou, Y. Jia, H. M. Wang, W. Q. Song, and P. Zou, "A P2P approach for global computing," *International Parallel and Distributed Processing Symposium 2003 (IPDPS 2003)*, pp. 6-11, Apr. 2003.

[44] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," *Chapter 11 in Grid Computing : Making the Global Infrastructure a Reality*, Wiley, 2003.

[45] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice : The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, issue 2-4, pp. 323-356, Feb. 2005.

[46] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor-A Distributed Job Scheduler," *Chapter 15 in Beowulf Cluster Computing with Linux*, The MIT Press, 2002.

[47] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and scheduling mechanisms for global computing applications," *The 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, pp.79-86, Apr. 2002.

[48] D. Kondo, M. Taufer, J. Karanicolas, C. L. Brooks, H. Casanova, and A. Chien, "Characterizing and Evaluating Desktop Grids: An Empirical Study," *The 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, pp. 26-35, Apr. 2004.

[49] D. Kondo, A. A. Chien, and H. Casanova, "Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," *The ACM/IEEE Conference on Supercomputing (SC2004)*, pp. 17-29, Nov. 2004.

[50] D. Kondo, "Scheduling Task Parallel Applications for Rapid Turnaround on Desktop Grids," *Ph.D. Thesis*, Department of computer Science and Engineering, University of California, San Diego, 2005.

[51] D. Kondo, B. Kindarji, G. Fedak, and F. Cappello, "Towards Soft Real-Time Applications on Enterprise Desktop Grids," *The Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006)*, pp. 65-72, May 2006.

[52] Korea@Home, http://www.koreaathome.org/eng/

[53] S. J. Choi, H. S. Kim, E. J. Byun, M. S. Baik, S. S. Kim, C. Y. Park, and C. S. Hwang, "Characterizing and Classifying Desktop Grid," *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2007), Sixth International Workshop on Global and Peer to Peer Computing (GP2P)*, pp. 743-748, May 2007.

[54] S. J. Choi, M. S. Baik, J. M. Gil, S. Y. Jung, and C. S. Hwang, "Adaptive Group Scheduling Mechanim using Mobile Agents in Peer-to-Peer Grid Computing Environment," *Applied Intelligence, Special Issue on Agent-based Grid Computing*, vol. 25, no. 2, pp. 199-221, Oct. 2006.

[55] S. J. Choi, M. S. Baik, J. M. Gil, C. Y. Park, S. Y. Jung, and C. S. Hwang, "Group-based Dynamic Computational Replication Mechanism in Peer-to-Peer Grid Computing," *IEEE/ACM International*

Symposium on Cluster Computing and the Grid (CCGRID 2006), Sixth International Workshop on Global and Peer to Peer Computing (GP2P), May 2006.

[56] S. J. Choi, M. S. Baik, J. M. Gil, C. Y. Park, S. Y. Jung, and C. S. Hwang, "Dynamic Scheduling Mechanism for Result Certification in Peer to Peer Grid Computing," *International Conference on Grid and Cooperative Computing (GCC 2005)*, LNCS 3795, pp.811-824, Dec. 2005.

[57] S. J. Choi, M. S. Baik, C. S. Hwang, J. M. Gil, and H. C. Yu, "Mobile Agent based Adaptive Scheduling Mechanism in Peer to Peer Grid Computing," *International Conference on Computational Science and its Applications (ICCSA 2005)*, LNCS 3483, pp. 936-947, May 2005.

[58] S. J. Choi, M. S. Baik, C. S. Hwang, J. M. Gil, and H. C. Yu, "Volunteer Availability based Fault Tolerant Scheduling Mechanism in Desktop Grid Computing Environment," *The 3th IEEE International Symposium on Network Computing and Applications, Workshop on Adaptive Grid Computing (NCA-AGC2004)*, pp.476-483, Aug. 2004.

[59] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing," *ACM Computing Surveys*, vol. 38, no. 1, pp. 1-53, Mar. 2006.

[60] M. Roehrig, W. Ziegler, and P. Wieder, "Grid Scheduling Dictionary of Terms and Keywords," *GFD-I.11, Grid Scheduling Dictionary WG (SD-WG)*, Nov. 2002

[61] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141-154, Feb. 1988.

[62] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing system," *IEE Proceedings Computers and Digital Techniques*, vol. 141, issue 1, pp. 1-10, Jan. 1994.

[63] T. D. Braun, H. J. Siegely, N. Becky, L. Boloniz, M. Maheswarany, A. I. Reuthery, J. P. Robertsony, M. D. Theysy, and B. Yaoy, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," *The 17th IEEE Symposium on Reliable Distributed Systems (SRDS 1998)*, pp. 330-335, Oct. 1998.

[64] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N.Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems," in *Advances in Computers: Volume 63: Parallel, Distributed, and Pervasive Computing*, vol. 63, pp. 93-129, Elsevier, Apr. 2005.

[65] I. Ekmecic, I. Tartalja and V. Milutinovic, "A survey of heterogeneous computing: concepts and systems," *Proceedings of the IEEE*, vol. 84, issue 8, pp. 1127-1144, Aug. 1996.

[66] S. Ali, T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous Computing," *Encyclopedia of Distributed Computing, J. Urbana and P. Dasgupta, eds.*, Kluwer Academic Publishers, Norwell, MA, 2002.

[67] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems, *The 8th Heterogeneous Computing Workshop (HCW'99)*, pp. 30-44, Apr. 1999.

[68] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, issue 6, pp. 810-837, Jun. 2001

[69] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, issue 2, pp. 135-164, Feb. 2002.

[70] J. Yu and R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *Special Issue on Scientific Workflows, SIGMOD Record*, vol. 34, no. 3, pp. 44-49, Sept. 2005.

[71] C. S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software: Practice and Experience*, vol. 36, issue 13, pp. 1381-1419, Nov. 2006.

[72] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing," *The First IEEE/ACM International Workshop on Grid Computing (Grid 2000)*, LNCS 1971, pp. 191-202, Dec. 2000.

[73] J. M. Shopf, "Ten actions when Grid scheduling," *Chapter 2 in Grid Resource Management: Sate of the Art and Future Trends*, Kluwer, 2003.

[74] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: a scalable lightweight architecture for high throughput computing," *The Fifth IEEE/ACM International Workshop on Grid Computing (GRID2004)*, pp. 19-25, Nov. 2004.

[75] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Y. Girard, R. Strachowski, and S. S. Yu, *Enabling Applications for Gird Computing with Globus*, Chapter 1,2,3,4, IBM Redbooks, Jun. 2003.

[76] R. Chow and T. Johnson, *Distributed Operating Systems & Algorithms*, Chapter 5, Addison-Wesley, 1997.

[77] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp.1327-1341, Sept. 1988.

[78] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *IEEE Computer*, vol. 25, no. 12, pp. 33-44, Dec. 1992.

[79] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *The 3rd Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1997)*, LNCS 1291, pp. 1-34, Apr. 1997.

[80] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman, "Reputation-Based Scheduling on Unreliable Distributed Infrastructures," *The 26th IEEE International Conference on Distributed Computing Systems (ICDCS2006)*, pp. 30-37, Jul. 2006.

[81] Y. Zhu, L. Xiao, Z. Xu, and L. M. Ni, "Incentive-based scheduling in Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 18, issue 14, pp. 1729-1746, Dec. 2006.

[82] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable Grid Computing," *The 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004)*, pp. 4-11, Apr. 2004.

[83] M. Fukuda, Y. Tanaka, N. Suzuki, and L. F. Bic, "A Mobile-Agent-Based PC Grid," *The 5th International Workshop on Active Middleware Services*, pp. 142-150, Jun. 2003.

[84] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," *The 9th Heterogeneous Computing Workshop*, pp. 349-363, May. 2000.

[85] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid Using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 4, pp. 369-382, Apr. 2003.

[86] J. Cao, A. T. Chan, Y. Sun, S.l K. Das, and M. Guo, "A taxonomy of application scheduling tools for high performance cluster computing," Cluster Computing, vol. 9, issue 3, pp. 355-371, Jul. 2006.

[87] S.J. Choi, M.S. Baik, H.S. Kim, E.J. Byun, and C.S. Hwang, "Reliable Asynchronous Message Delivery for Mobile Agent," *IEEE Internet Computing*, vol. 10, issue 6, pp. 16-25, Dec. 2006.

[88] S.J. Choi, M.S. Baik, and C.S. Hwang, "Location Management & Message Delivery Protocol in Multi-region Mobile Agent Computing Environment," *The 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pp. 476-483, Mar. 2004.

[89] ODDUGI mobile agent system, http://oddugi.korea.ac.kr/

[90] P. Maes, R. H. Guttman, and A. G. Moukas, "Agents That Buy and Sell," *Communications of the ACM*, Vol. 42, No. 3, pp. 81-91, Mar. 1999.

[91] D. Wong, N. Paciorek, and D. Moore, "Java-based Mobile Agents," *Communication of the ACM*, Vol. 42, No.3, pp.92-102, Mar. 1999.

[92] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," *The 2nd International Workshop on Peer-to-Peer Systems*, LNCS 2735, pp. 256-267, Feb. 2003.

[93] Y. Li and M. Mascagni, "Improving Performance via Computational Replication on a Large-Scale Computational Grid," *The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 442-448, May 2003.

[94] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," *The 2nd International Workshop on Grid Computing*, LNCS 2242, pp.75-86, Nov. 2001.

[95] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities," *The 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 346-351, May 2002.

[96] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," *The Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, pp. 177-190, Aug. 2002.

[97] F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen, "Autonomous Replication for High Availability in Unstructured P2P systems," *The 22nd International Symposium on Reliable Distributed Systems*, pp. 99-108, Oct. 2003.

[98] C. G. Renaud and N. Playez, "Result Checking in Global Computing Systems," *The 17th Annual ACM International Conference on Supercomputing (ICS'03)*, pp. 226-233, Jun. 2003.

[99] P. Jalote, *Fault Tolerance in Distributed Systems*, Prentice-Hall, 1994.

[100] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.

[101] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd Edition, WILEY, 2002.

[102] Yu. A. Zuev, "On the Estimation of Efficiency of Voting Procedures", *Theory of Probability & Its Applications*, vol. 42, no. 1, pp. 73-81, 1998.

Acknowledgements

I would like to thank many people who helped make this thesis become a reality. I have received huge support and help throughout this long journey.

I would like to express my deepest gratitude towards my advisor, professor ChongSun Hwang, for his invaluable help and advice throughout my 6 years at Korea University. Especially, he gave me the principle that guides my life, "Love yourself, be humble, show consideration for others, and smile cheerfully".

I would also like to thank my thesis committee, SangKeun Lee, Myong-Soon Park, YooHun Won, DukHoon Kwak, and HeonChang Yu for their valuable time and comment.

I would like to express my gratitude to the members in Distributed Systems Laboratory (*disysers*), Korea University. They always helped me a lot. Especially, I am indebted to MaengSoon Baik for his endless support and encouragement. He gave me not only invaluable help and technical advice, but also much friendship, cheer and consideration. I owe special thanks to the members of the Fault Tolerance Group for providing me with much inspiration and encouragement. I would like to express my sincere gratitude to JoonMin Gil, HongSoo Kim, and EunJoung Byun for their help and thoughtfulness. I was very happy and fortunate to work with them. Their support and encouragement throughout the ups and downs in graduate school life has been my source of patience and endurance.

Finally, and most specially, I would like to thank my parents, brothers, sisters, sisters-in-law, brothers-in-law, nephews, and nieces. Without their unconditional love, care, and encouragement, I would not be where I am now. I am grateful for everything they have given me.

All of the work done for this thesis would not have been possible without all of them. I am what I am thanks to their efforts. Words cannot express the gratitude I have. This thesis is dedicated to all of them.