A Taxonomy of Desktop Grids and its Mapping to State-of-the-Art Systems

SUNGJIN CHOI, RAJKUMAR BUYYA
University of Melbourne, Australia
and
HONGSOO KIM, EUNJOUNG BYUN
Korea University, Korea
and
MAENGSOON BAIK
Samsumg SDS, Korea
and
JOONMIN GIL
Catholic University of Daegu, Korea
and
CHANYEOL PARK
Supercomputing Center, KISTI, Korea

Desktop Grid has emerged as an attractive computing paradigm for high throughput applications. In Desktop Grid systems, numerous desktop computers owned by different individuals are employed as computational resources at the edge of the Internet. Accordingly, building such systems is complicated due to resources' heterogeneity, failures, non-dedication, volatility and lack of trust. Therefore, it is important to comprehend how these distinct characteristics impact on architecture, execution model, resource management, and scheduling. In this article, architectural elements are investigated and then a new taxonomy of Desktop Grids is proposed. The taxonomy puts emphasis on: (a) system, (b) application, (c) resource, and (d) scheduler perspectives. Then, the proposed taxonomy is mapped to various state-of-the-art systems to identify their unique elements, strengths, weaknesses, and challenges. Based on these mapping studies, a gap analysis is performed and the future directions of Desktop Grids are proposed, which help with better design and development of new systems and scheduling algorithms.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed systems—*distributed applications*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms: Design, Algorithm, Management

Additional Key Words and Phrases: Desktop Grid, scheduling, resource management, taxonomy

Author's address: SungJin Choi, Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, 5.21/111 Barry Street, Carlton, Victoria 3053, Australia; email:lotieye@csse.unimelb.edu.au

1. INTRODUCTION

Grid computing has recently emerged as a promising paradigm for high performance or high throughput applications because of the vast development of powerful computers and high-speed network technologies as well as their low cost [Berman et al. 2003; Foster and Kesselman 2004; Li and Baker 2004; Baker et al. 2002; Foster and Iamnitchi 2003]. It aims to aggregate heterogeneous, large-scale and multiple-institutional resources, and to provide transparent, secure and coordinated access to various computing resources (for example, supercomputer, cluster, scientific instrument, database, storage, etc.) owned by multiple institutions by creating virtual organization [Berman et al. 2003; Foster and Kesselman 2004; Li and Baker 2004; Baker et al. 2002; Foster and Iamnitchi 2003]. On the other hand, Desktop Grid computing[1] aims to harvest a number of idle desktop computers owned by individuals at the edge of the Internet [Foster and Iamnitchi 2003; BOINC ; Fedak et al. 2001; Chien et al. 2003; Sarmenta 2001; Kondo 2005; Choi et al. 2007]. It has recently gained rapid interest and attraction because of the success of the most popular examples such as GIMPS [GIMPS], distributed.net [Distributed.net] and SETI@Home [SETI@home].

Desktop Grid computing is a type of Grid computing, but there are several distinct differences between them in terms of the type and characteristics of resources and the type of sharing (see Table I). Building such systems, especially offering enterprise class services, is complicated due to resources' heterogeneity, failures, non-dedication, volatility and lack of trust, since they are at the edge of the Internet and owned by individuals [Milojicic et al. 2002; Barkai 2002; Steinmetz and Wehrle 2005; Subramanian and Goodman 2005; BOINC ; Anderson 2004; Taufer et al. 2005; Fedak et al. 2001; Chien et al. 2003; Sarmenta 2002; Zhou and Lo 2005; Zhao and Lo 2005; Kondo et al. 2002; Kondo et al. 2004; Choi et al. 2007; Choi et al. 2006]. Particularly, resource management and scheduling are different from Grid in terms of process, organization and goal (see section 2.2). These distinct characteristics have a direct effect on system performance and reliability. If Desktop Grid systems do not consider non-dedication, volatility and lack of trust of volunteers, performance and reliability will be seriously deteriorated. Therefore, it is important to comprehend how these distinct characteristics impact on architecture, execution model, resource management and scheduling.

This article aims to identify and comprehend concepts, architecture, execution model and characteristics to be considered when developing and implementing Desktop Grid systems. In this article, a new taxonomy of Desktop Grids is provided focusing on: (a) system, (b) application, (c) resource and (d) scheduler perspectives. The taxonomy considers various application's requirements, resource's properties and scheduling algorithms when analyzing and classifying Desktop Grid systems. Then, the proposed taxonomy is mapped to various state-of-the-art systems to identify their unique elements, strengths, weaknesses and challenges. Based on these mapping studies, a gap analysis is performed and the future directions of Desktop

[1]Desktop Grid computing is also called volunteer computing [BOINC ; Sarmenta 2001], global computing [Fedak et al. 2001; Neary et al. 2000; Dou et al. 2003; Kondo et al. 2002], Peer-to-Peer Grid computing [Zhou and Lo 2005; Choi et al. 2006], public-resource computing [Anderson 2004] and Peer-to-Peer cycle sharing systems [Zhao and Lo 2005].

Grids are proposed, which help with better design and development of new systems and scheduling algorithms.

This article has several novel contributions towards the improvement of the understanding of Desktop Grid and the advance of the area of resource management and scheduling in regard to Desktop Grid systems. In addition, it aims to help comprehend the definition, architecture, execution model, characteristics and applications of Desktop Grid. It also aims to help characterize and classify resource management and scheduling mechanisms, and furthermore develop new ones.

The rest of the article is structured as follows. Section 2 presents the overview of Desktop Grid and its differences with Grid. Section 3 describes the design issues and layered architectural elements. Section 4 presents a new taxonomy of Desktop Grids focusing on system, application, resource and scheduler perspectives. Section 5 presents a mapping to state-of the-art-systems. Section 6 presents gap analysis, challenging issues and the future directions. Section 7 illustrates related studies on taxonomy of Grids and Desktop Grids. Finally, conclusions are given in Section 8.

2. OVERVIEW

2.1 Desktop Grid

Desktop Grid aims to harvest a number of idle desktop computers (that is, volunteers) owned by individuals at the edge of the Internet [Foster and Iamnitchi 2003; BOINC ; Fedak et al. 2001; Chien et al. 2003; Sarmenta 2001; Kondo 2005; Choi et al. 2007]. A Desktop Grid computing environment [2] mainly consists of *client*, *volunteer* and *server*, as shown in Figure 1. A ***client*** is a parallel job submitter who requests for results. 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. It can have a file server to maintain tasks and results files. 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 the final result of the job back to the client.

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. Volunteers have different execution behavior. In addition, some malicious volunteers may tamper with their computations

[2]The services of client and server can be performed by one machine or the services of server and volunteer can be conducted by one machine according to architecture, organization, and model of Desktop Grid systems

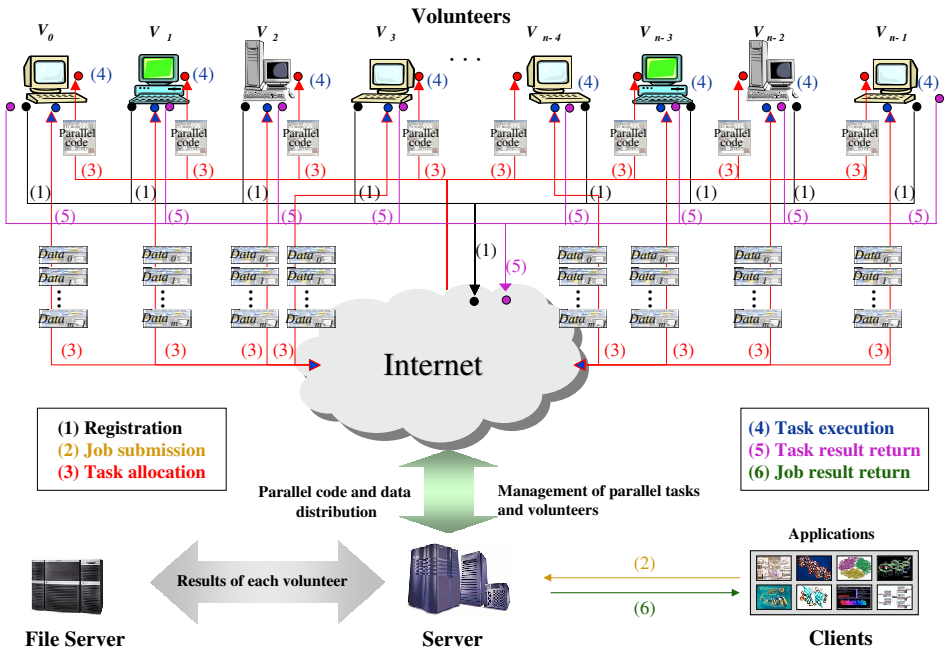

Fig. 1. Desktop Grid computing environment

and return corrupt results. A variety of hardware and software of volunteers lead to deviation from the result of a task. These distinct features make it difficult for a server to schedule tasks and manage the allocated tasks and volunteers. Consequently, they cause the delay and blocking of the execution of tasks and the partial or entire loss of the executions, and result in performance degradation.

Desktop Grid systems usually support embarrassingly parallel applications, which consist of many instances of the same computation each with its own data [Berman et al. 2003; Foster and Kesselman 2004; Anderson 2004; Fedak et al. 2001; Chien et al. 2003; Sarmenta and Hirano 1999; Neary et al. 1999; Baratloo et al. 1999; Korea@Home]. The applications are usually involved with scientific problems that need large amounts of processing capacity over long periods of time. Desktop Grid computing has been utilized in a variety of fields: mathematics, cryptography, high energy physics, molecular biology, medicine, astrophysics, climate study, chemistry, and so on [GIMPS ; Distributed.net ; SETI@home ; BOINC ; Anderson 2004; Korea@Home]. Examples are **GIMPS** which searches for Mersenne prime numbers, **distributed.net** which finds a solution for the RSA secret-key challenge, **SETI@Home** which detects intelligent life outside Earth, **Climateprediction.net, BBC Climate Change and Korea@Home** which study climate change, **fightAIDS@Home** which discovers new drugs to cure diseases and cancers, **Folding@Home, Predictor@Home, Genome@Home, Korea@Home and Docking@Home** which explore protein structures & sequences and the physical processes of protein folding, **LHC@Home** which simulates particles travelling

around the LHC (Large Hadron Collider), and **Einstein@Home** which detects gravitational wave.

Some studies have been made on Desktop Grid systems, which provide an underlying platform: Alchemi [Luther et al. 2005], Bayanihan [Sarmenta and Hirano 1999; Sarmenta 2002; 2001], BOINC [BOINC ; Anderson 2004; Taufer et al. 2005; Anderson et al. 2005], Cluster Computing On the Fly(CCOF) [Zhou and Lo 2004; Lo et al. 2004; Zhou and Lo 2005; Zhao and Lo 2005], Charlotte [Baratloo et al. 1999], Condor [Thain et al. 2003; 2005; Tannenbaum et al. 2003], Computer Power Market (CPM) [Buyya and Vazhkudai 2001; Ping et al. 2001], Entropia [Chien et al. 2003; Chien et al. 2003], Javelin [Neary et al. 1999; Neary et al. 2000; Neary and Cappello 2005], Korea@Home [Korea@Home ; Choi et al. 2007; Choi et al. 2006; Choi et al. 2006; 2005; Byun et al. 2007; Choi et al. 2004], Messor [Babaoglu et al. 2002; Montresor et al. 2003], Organic Grid [Chakravarti et al. 2005; 2006], Paradropper [Zhong et al. 2003; Dou et al. 2003], POPCORN [Nisan et al. 1998], WebCom [Morrison et al. 2001; 2002], XtremWeb [Fedak et al. 2001; Cappello et al. 2005], and so on.

2.2 Desktop Grid vs. Grid

Desktop Grid has recently become appealing for executing high throughput applications, because CPU, storage and network capacities become more advanced and cheaper. It is different from Grid in terms of the types and characteristics of resources, and the types of sharing [Berman et al. 2003; Foster and Kesselman 2004; Foster and Iamnitchi 2003; Milojicic et al. 2002; Barkai 2002; Subramanian and Goodman 2005; Anderson 2004; Chien et al. 2003; Sarmenta 2001; Choi et al. 2007; Choi et al. 2006] (see Table I). The resources of Desktop Grid are mainly personal computers (that is, desktop), whereas Grid resources include supercomputer, cluster, scientific instrument, database, storage, etc. They are highly-volatile, non-dedicated and highly-heterogeneous, much different from that of Grid. They are also more malicious, unreliable and faulty than Grid resources. Desktop Grid resources are administrated by individual users, whereas Grid resources are managed by professional administrators. The applications of Desktop Grid mainly have no dependency between tasks, so it is not necessary to communicate between volunteers. In the case of institution-based Desktop Grid (refer to Section 4 for the definition), dependent applications (for example, workflow applications) can be applicable. On the other hand, Grid deals with dependent applications (for example, workflow or MPI applications) as well as independent ones, so communication between nodes often happens. 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 runs).

Scheduling is one of the most challenging problems in Grid computing in general, and Desktop Grid in particular. It 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) [Roehrig et al. 2002; Casavant and Kuhl 1988; Rotithor 1994; Ali et al. 2002; Maheswaran et al. 1999; Shopf 2003]. 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

Table I. A comparison of Grid and Desktop Grid

Items	Desktop Grid (DG)		Grid
	Volunteer DG[1] (Internet-based)	Enterprise DG[1] (LAN-based)	
Resource	Desktop * Anonymous volunteers	Desktop * within a corporation, university, institute, etc.	Supercomputer, cluster, scientific instrument, database, storage * Virtual organization (VO)
Connection	-Non-dedicated and poor bandwidth -Immediate presence (connectivity) -Consider firewall, NAT, Dynamic address	-Non-dedicated and intermediate bandwidth -More constant connectivity than Volunteer DG	Dedicated and high speed bandwidth
Heterogeneity	High heterogeneity	Intermediate heterogeneity * Less heterogeneous than Volunteer DG	Low heterogeneity
Dedication	-Non-dedicated -High volatility * Need an incentive mechanism	-Non-dedicated (mainly) -Dedicated (possible) -Low volatility (non-business hours) * Need an incentive mechanism	Dedicated * Is able to use reservation
Trust	Malicious volunteer * Need result certification	Low trustworthy resources	High trustworthy resources
Reliability	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 Model	-Independent (mainly) -Computation-intensive (mainly) -High-throughput (mainly)	-Independent (mainly) -Dependent[2] (possible) (e.g., workflow applications) -Computation-intensive (mainly) * Data-intensive (possible) -High throughput (mainly)	-Independent -Dependent[2] (e.g., workflow or MPI applications) -Computation or data-intensive -High performance (mainly)
Inter-node communication	No communication between volunteers	-No communication (mainly) -Low communication (applicable) (e.g., workflow applications)	-Low communication between nodes (e.g., workflow application) -High communication between nodes (e.g., MPI applications)

1. Refer to Section 4 for the definition of Volunteer or Enterprise DG.
2. In Enterprise DG, 'dependent' means flow-dependency, whereas 'dependent' means both flow- and execution-dependency in Grid (Refer to Section 5.1).

monitoring of task execution are involved because of the heterogeneous and dynamic nature of Grid resources. On the other hand, ordering tasks focuses on placing priority on tasks to be executed at a specific node or site.

Grid systems generally perform scheduling in a hierarchical manner [Krauter et al. 2002; Hamscher et al. 2000; Jacob et al. 2003]. In other words, a Grid scheduler consists of a meta-scheduler (or a super scheduler) and local schedulers. 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 [Roehrig et al. 2002; Krauter et al. 2002; Hamscher et al. 2000; Jacob et al. 2003]. LoadLeveler, LSF, or PBS can be used as a local scheduler [Roehrig et al. 2002; Krauter et al. 2002; Hamscher et al. 2000; Jacob et al. 2003].

Desktop Grid scheduling is different from Grid scheduling because Desktop Grid is different from Grid in terms of the type of resource, dedication, trust, reliability, application model, and so on [Foster and Iamnitchi 2003; Anderson 2004; Chien et al. 2003; Sarmenta 2001; Kondo et al. 2004; Choi et al. 2007; Choi et al. 2006] (see Table I). First, Desktop Grid scheduling mainly focuses on the process of assigning tasks to the most suitable resources (that is, to decide where to execute tasks) [Anderson et al. 2005; Cappello et al. 2005; Chien et al. 2003; Sarmenta 2001; Neary and Cappello 2005; Baratloo et al. 1999; Zhou and Lo 2005; Chakravarti et al. 2005; Montresor et al. 2003; Kondo et al. 2004; Kondo 2005; Choi et al. 2006]. It can be performed in a centralized way or in a fully distributed way. Second, unlike Grid, most Desktop Grid systems do not need a local scheduler in that a scheduling unit is a single desktop computer, not a site in Grid. Finally, Desktop Grid scheduling is opportunistic. Desktop Grid respects the autonomy of volunteers (that is, volunteers can freely participate in or leave public execution). Thus, Desktop Grid scheduling should use resources as quickly as possible when the resources are available or idle [BOINC ; Sarmenta and Hirano 1999; Baratloo et al. 1999; Thain et al. 2005; Choi et al. 2006].

Resource management of Desktop Grid is also different from that of Grid. Desktop Grid computing is complicated by heterogeneous, volatile, faulty and malicious resources. Therefore, it should focus more on non-dedication, volatility, lack of trust and heterogeneous properties than Grid [Sarmenta 2002; Zhou and Lo 2005; Zhao and Lo 2005; Kondo et al. 2002; Kondo et al. 2004; Choi et al. 2007; Choi et al. 2006; Choi et al. 2006; 2005; Choi et al. 2004; Sonnek et al. 2007]. For example, Desktop Grid systems should focus more on resource grouping in order to execute tasks and manage volunteers efficiently. It enables a scheduler to apply appropriate scheduling algorithms to each group that have similar execution behaviors. Desktop Grid systems should also provide result certification to tolerate malicious volunteers. Furthermore, they should consider reputation and incentive mechanisms to encourage volunteers to donate their resource eagerly, reliably and trustworthily. Desktop Grid scheduling can be couple with them to give more benefits and rewards to eager, reliable and trustworthy volunteers.

3. ARCHITECTURE

3.1 Design Issues

Desktop Grid systems should have the following requirements: security, reliability, trust, performance, unobtrusiveness & opportunism, scalability and incentive [Anderson 2004; Fedak et al. 2001; Chien et al. 2003; Sarmenta and Hirano 1999; Neary et al. 1999; Baratloo et al. 1999; Korea@Home ; Abbas 2003].

—**Security**: Desktop Grid systems should protect the integrity of volunteers. They must prevent the running tasks from accessing or modifying files or data on volunteers. Security is fundamental to encourage participation in Desktop Grid computing.

—**Reliability**: A reliable execution should be guaranteed. Desktop Grid systems should tolerate the volunteer's failures such as crash failure, network failure and volatility.

—**Trust**: It is important to guarantee the correctness of results. Desktop Grid systems should tolerate erroneous results generated by malicious volunteers or variation due to heterogeneous hardware/software versions.

—**Incentive**: Desktop Grid systems should provide an incentive method in order to encourage eager participation. It evaluates and ranks volunteers according to their execution behavior. According to the assessment and ranking, it gives volunteers benefits (that is, rewards). Furthermore, Desktop Grid systems should provide incentive/reputation-based scheduling, which distributes more tasks to eager and reliable volunteers, and consequently gives more reputation and credit to them.

—**Unobtrusiveness and Opportunism**: Volunteers are voluntary participants, so Desktop Grid has to respect the autonomy of volunteers. Volunteers can join and leave their public execution at their own will. As soon as desktop owners use their computers, the running public execution should be stopped immediately and the resources should be yielded to private execution. When desktop computers are available, Desktop Grid systems should employ idle resources as quickly as possible.

—**Performance**: It is important how fast or how many tasks are completed. Desktop Grid systems should have an efficient scheduling mechanism for better performance. Particularly, it should adapt to a dynamic, heterogeneous and unreliable environment.

—**Scalability**: Desktop Grid systems should be scalable and able to manage volunteers without deteriorating performance even if the number of volunteers grows or even if volunteers are spread across the Internet. Distributed algorithms (for example, distributed scheduling and resource management algorithms) can be used instead of centralized algorithms for better scalability.

—**Ease of Use and Deployment**: Desktop Grid is based on the voluntary users who do not have the professional skills of Grid technology. If sophisticated skills are required during installation and deployment, the spread of participation in Desktop Grid computing is hindered. For more users to join in Desktop Grid, it is guaranteed that they are able to easily install and run the system without any

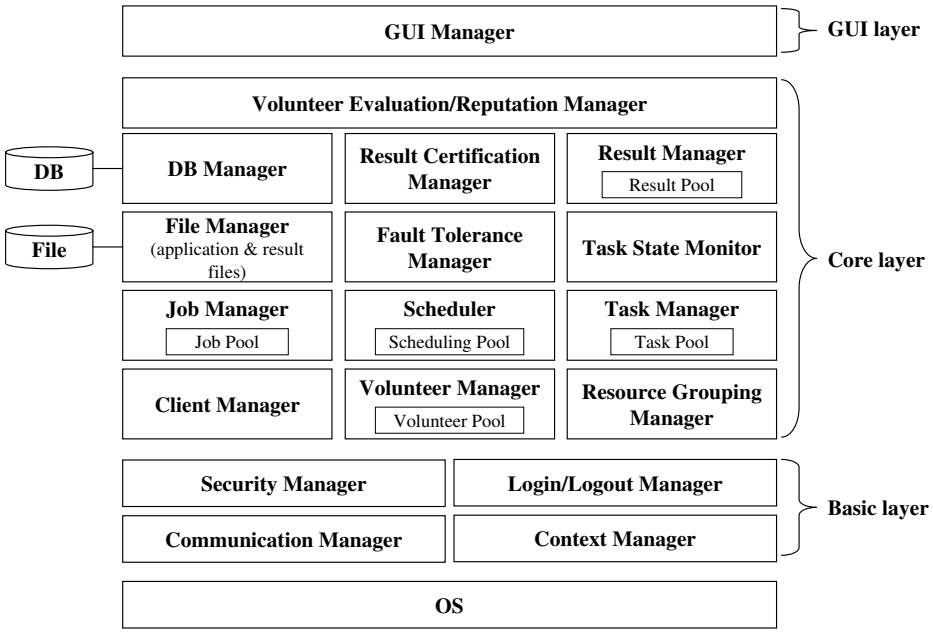

Fig. 2. Server architecture

requirement of professional skills. In addition, the system should provide GUI (Graphic User Interface) not only for users to control the submitted tasks, but also for administrators to monitor and manage the volunteers and the allocated tasks.

3.2 Layered Architecture

The architectures of server, volunteer and client are described as shown in Figures 2, 3 and 4 .

3.2.1 *Server.* A server's components can be organized in a layered architecture: basic, core and GUI layers, as shown in Figure 2.

The basic layer provides basic functionalities such as configuration, communication and security. These are also common to client and volunteer. It consists of context, communication, security and login/logout managers.

—**Context manager**: It configures the server system.
—**Communication manager**: It is responsible for interactions between server and client, or between server and volunteer.
—**Security manager**: It provides secure access to resources and secure communication among server, client and volunteer.
—**Login/logout manager**: It handles registration or participation processes of public execution.

The core layer provides main functionalities such as scheduling, management of volunteers and clients and control of jobs and tasks.

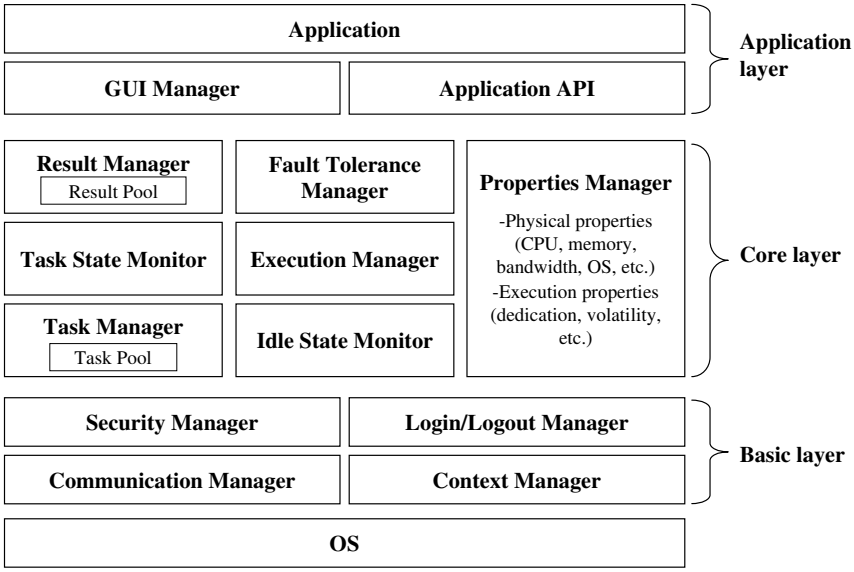

Fig. 3. Volunteer architecture

—**Client manager**: It manages clients who submit jobs to the server.

—**Volunteer manager**: It is responsible for managing volunteers. It maintains the volunteer's properties such as capabilities (CPU, OS type, memory, network, etc.), dedication, volatility and credibility. It also maintains the list of volunteers who participate in public execution.

—**Resource grouping manager**: It forms volunteer groups according to the properties of volunteers.

—**Job manager**: It controls jobs submitted by clients. It also splits a job into tasks.

—**Scheduler**: It is responsible for scheduling. It also maintains scheduling information in a scheduling pool. It collaborates with various managers such as volunteer, resource grouping, job, task, fault tolerance, result certification, DB and volunteer evaluation/reputation managers, to achieve its own scheduling goals and purposes.

—**Task manager**: It manages tasks distributed to volunteers. It is also responsible for transferring tasks (code and data) to volunteers, cooperating with the file manager.

—**Task state monitor**: It monitors the status of tasks that volunteers are executing. It cooperates with the task state monitor of a volunteer.

—**File manager**: It controls application and result files. Particularly, it cooperates with the task or job managers in transferring large or huge data reliably and effectively.

—**Fault tolerance manager**: It handles the failures of volunteers which occur in

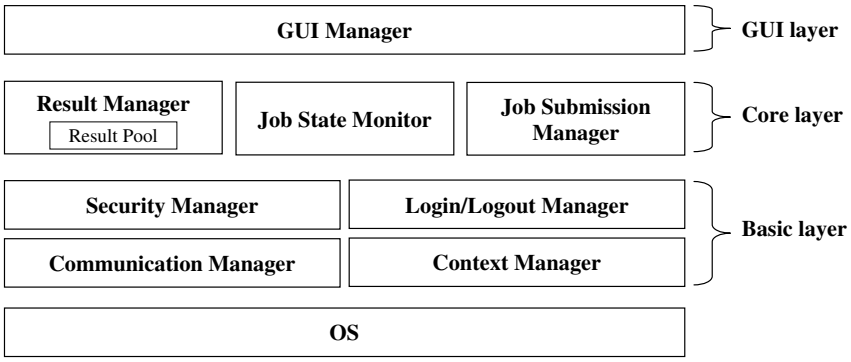

Fig. 4. Client architecture

the middle of public execution. It cooperates with the scheduler in rescheduling tasks.

—**DB manager**: It keeps and manipulates a lot of information related with volunteers, clients, tasks, scheduling on a database.

—**Result certification manager**: It detects and tolerates the erroneous results generated by malicious volunteers or the deviated results due to a variety of hardware and software.

—**Result manager**: It manages task results returned by volunteers. It cooperates with the file manager in transferring result files.

—**Volunteer evaluation/reputation manager**: It evaluates volunteers, and scores and ranks them according to their execution behavior and history.

—**Incentive manager**: It encourages volunteers to donate their resources eagerly, reliably and trustworthily by giving incentives such as money, opportunity to use resources, or scheduling priority, ranking, etc.

The GUI layer provides a graphical interface, with which an administrator can control the server more easily.

—**GUI manager**: It provides a graphic interface to users, developers, or administrators.

3.2.2 *Volunteer.* The volunteer's components can be organized in a layered architecture: basic, core and application layers, as shown in Figure 3. The basic layer has the same functionalities as the server. Particularly, the security manager is also responsible for secure execution of tasks.

The core layer provides main functionalities such as task execution and management.

—**Task manager**: It manages tasks transferred from a server.

—**Task state monitor**: It monitors the status of tasks. It reports the status to the task state monitor in its server.

—**Idle state monitor**: It checks if its CPU is idle. It notifies the execution manager of the status.

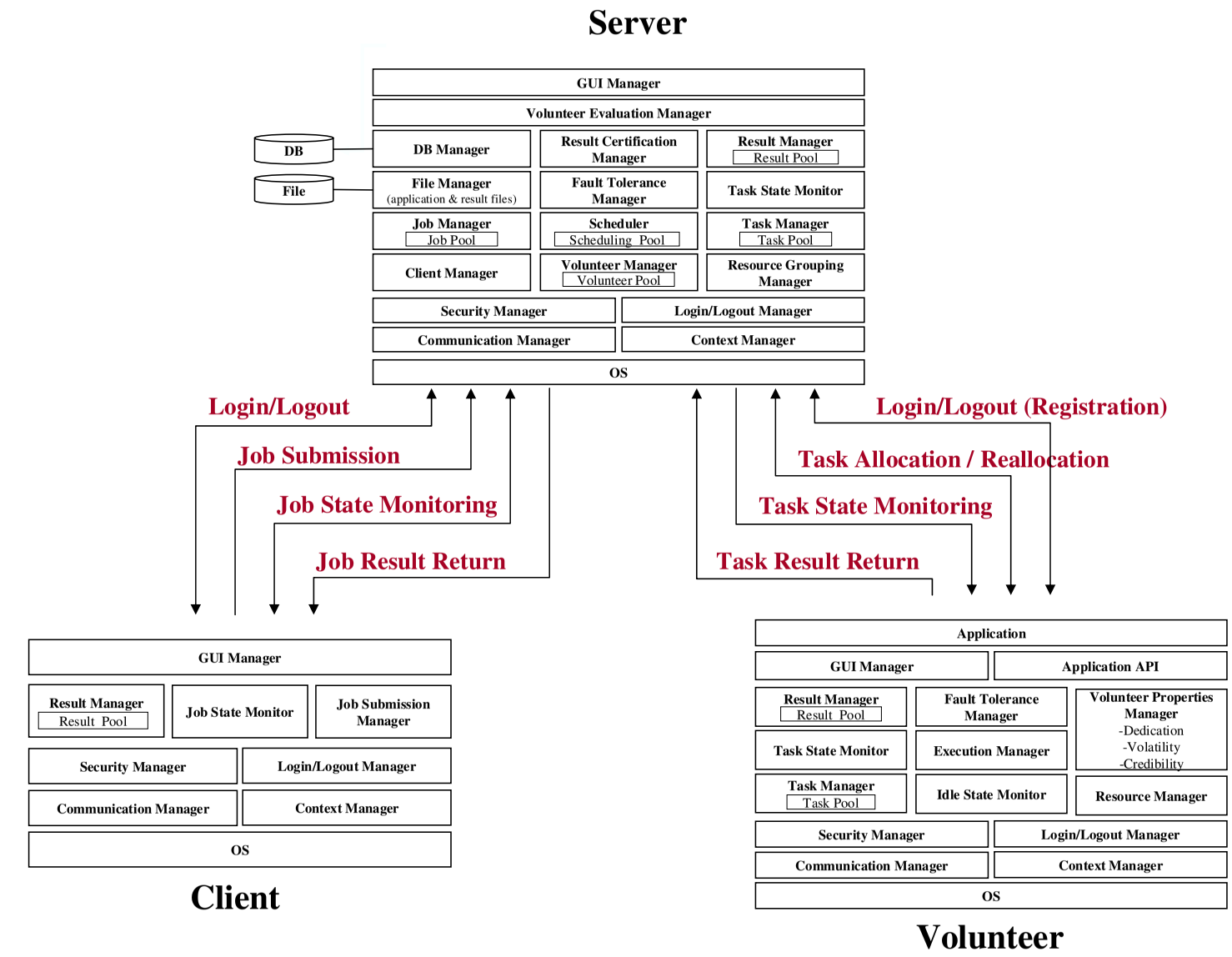

Fig. 5. Interaction among server, client and volunteer.

—**Properties manager**: It investigates, checks and maintains volunteer's hardware and software properties (that is, physical properties). It also checks and manages volunteer's properties related to public execution (that is, execution properties), such as dedication, volatility, and so on.

—**Execution manager**: It executes tasks allocated by its server. It controls (that is, starts, restarts, stops) the execution depending on the status notified by the idle state monitor.

—**Fault tolerance manager**: It detects and controls the failures that occur in the middle of public execution. It notifies the execution manager as well as the fault tolerance manager in a server side.

—**Result manager**: It returns the task result. It cooperates with the result manager in a server side.

The application layer provides a graphical interface, by which a user can control one's resource and public execution more easily. It also provides application API for developers.

—**GUI manager**: It provides a graphic interface to users.

—**Application API**: It provides API (application programming interface) for application developers. With the API, application developers can develop applications running on the middleware (or system).

—**Application**: Applications are implemented on the basis of Application API.

3.2.3 *Client.* The client's components can be organized in a layered architecture: basic, core and GUI layers, as shown in Figure 4. The basic and GUI layers have the same functionalities as the server. The core layer consists of result manager, job state monitor and job submission manager.

—**Result manager**: It manages job's results returned from a server.

—**Job state monitor**: It monitors the status of jobs that a server is processing.

—**Job submission manager**: It submits jobs to a server. It cooperates with the job manager in the server.

3.2.4 *Interactions among Server, Client and Volunteer.* A server interacts with clients by exchanging messages such as login / logout, job submission, job state monitoring and job result return as shown in Figure 5. In addition, a server interacts with volunteers by exchanging messages such as registration, task allocation, task state monitoring, and task result return as shown in Figure 5.

4. TAXONOMY OF DESKTOP GRIDS

A new taxonomy of Desktop Grid systems is presented focusing on (a) system, (b) application, (c) resource and (d) scheduler perspectives as shown in Figure 6.

4.1 System Perspective

Desktop Grids are investigated and analyzed with the system perspectives. Desktop Grids are categorized according to organization, platform, scale and resource properties (see Figure 7).

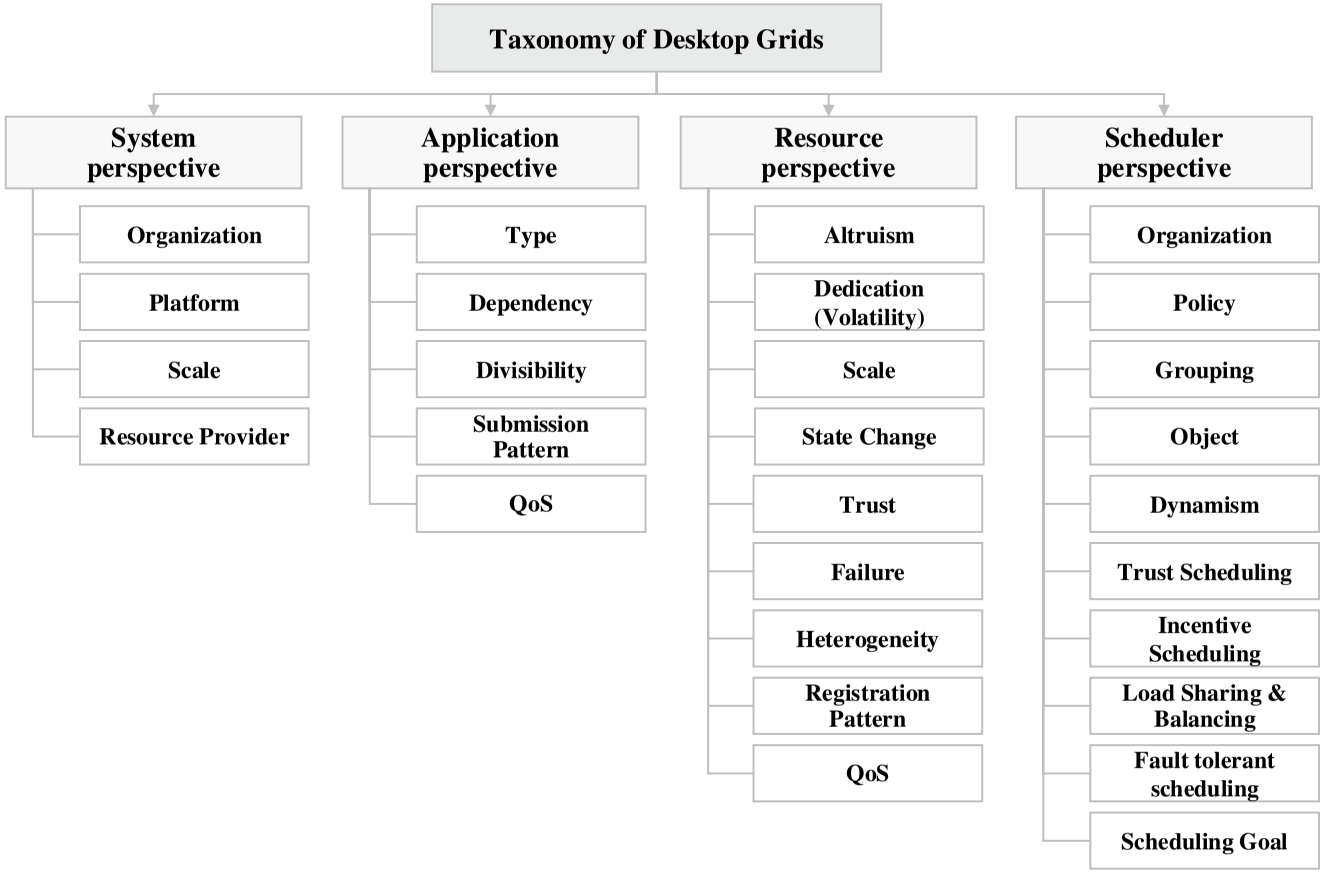

Fig. 6. A taxonomy of Desktop Grids (System Perspective)

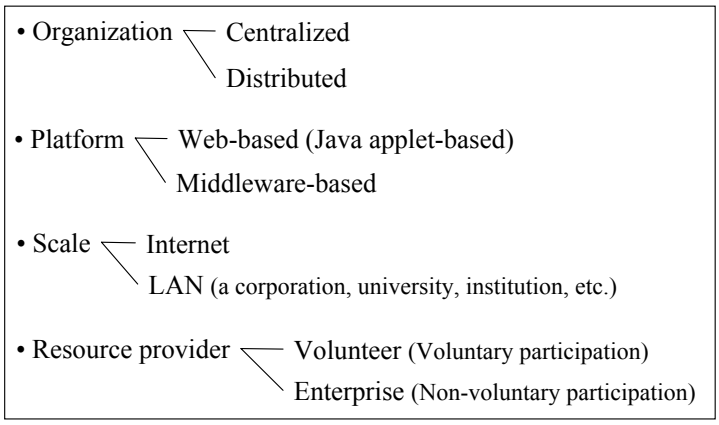

Fig. 7. A taxonomy of Desktop Grid (System's Perspective)

4.1.1 ***Organization.*** Desktop Grids are categorized into two types: centralized and distributed, according to the organization of components.

—**Centralized Desktop Grid**: Centralized Desktop Grid (DG) consists of client, volunteer and server. The execution model of centralized DG 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 8. Typical examples are BOINC, XtremWeb, Entropia, Bayanihan, Korea@Home, and so on.

(1) *Registration phase*: Volunteers register their information to a server.
(2) *Job submission phase*: A client submits a job to a server.
(3) *Resource grouping phase* [3]: A server constructs volunteer groups according to capability, availability, reputation, trust, and so on [Chien et al. 2003; Choi et al. 2006; Choi et al. 2006; 2005]. Scheduling is performed on the basis of groups [Choi et al. 2006; Choi et al. 2006; 2005].
(4) *Task allocation phase*: A server distributes tasks to the registered volunteers by means of scheduling algorithms.
(5) *Task execution phase*: Each volunteer executes its task.
(6) *Task result return phase*: Each volunteer returns the result of its task to the server.
(7) *Result certification phase*: The server checks the correctness of the returned results in order to tolerate malicious volunteers [Sarmenta 2002; Choi et al. 2005; Renaud and Playez 2003], or to deal with variations in numerical processing due to a variety of hardware and software [Taufer et al. 2005].
(8) *Job result return phase*: The server returns the final result of the job to the client.

[3]The resource grouping phase makes scheduling more efficient, which enables a scheduler to apply various scheduling policies, fault tolerance, and result certification algorithms to each group, or make resource management easier. However, most of the centralized DG systems do not provide this phase.

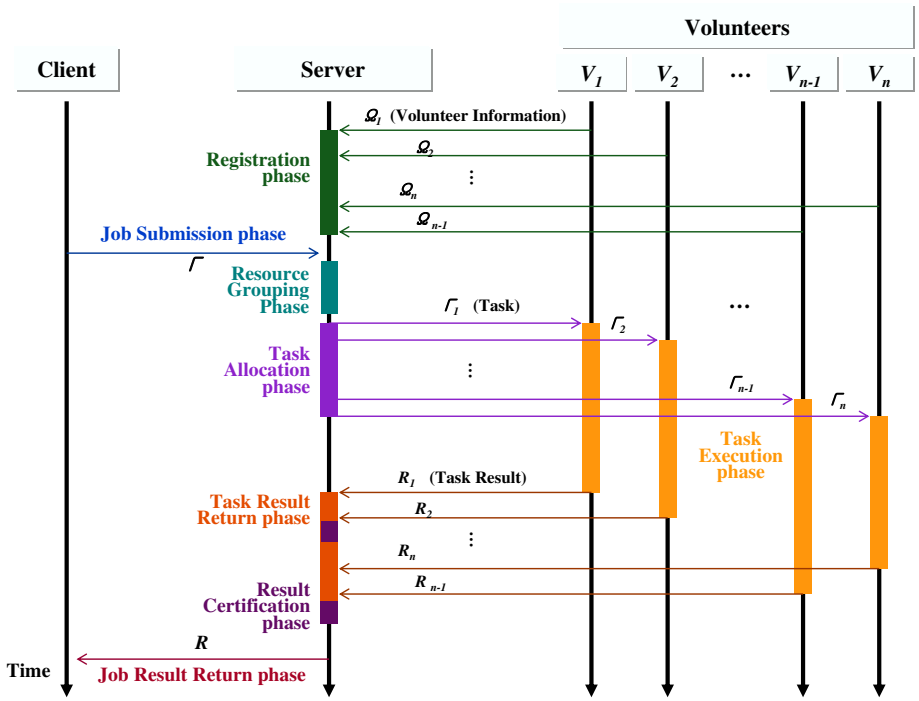

Fig. 8. Execution model of centralized Desktop Grid

—**Distributed Desktop Grid**: Distributed Desktop Grid[4] consists of client and volunteer. In contrast to centralized DG, there is no server; therefore volunteers take the place of the server [5]. For example, volunteers maintain the partial information of other volunteers. They are also responsible for scheduling. The execution model of distributed DG consists of eight phases: registration, job submission, computational overlay network (CON) construction, task allocation, task execution, task result return, result certification, and job result return phase as shown in the Figure 9. Typical examples are CCOF, Organic Grid, Messor and Paradropper.

(1) *Registration phase*: Volunteers exchange their information between other volunteers.[6]

(2) *Job submission phase*: A client consigns a job to its neighbor volunteers.

[4]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 [Milojicic et al. 2002; Barkai 2002; Steinmetz and Wehrle 2005; Subramanian and Goodman 2005]. 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 [Choi et al. 2006; Choi et al. 2006; 2005].

[5]A client or a broker can take the place of the server.

[6]Volunteers can register their information to brokers [Neary et al. 1999; Buyya and Vazhkudai 2001].

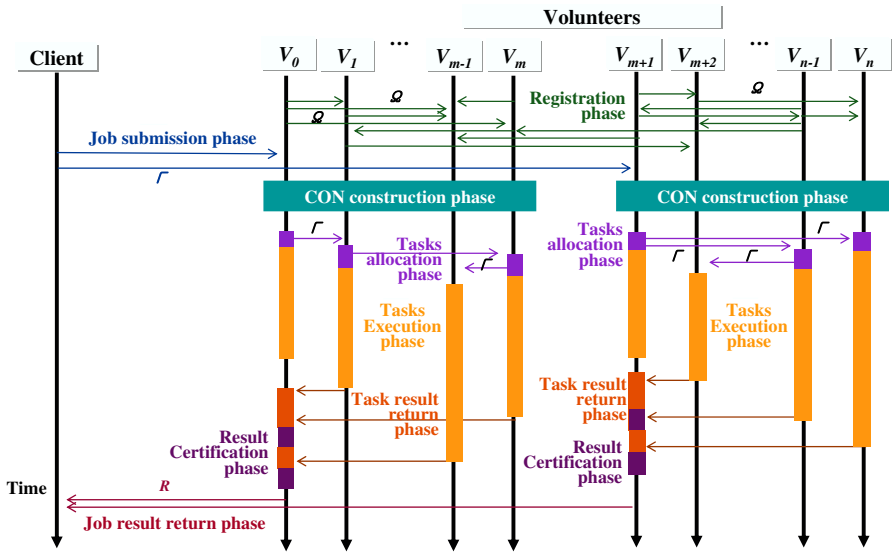

Fig. 9. Execution model of distributed Desktop Grid

(3) *CON construction phase*: Volunteers self-organize their CON according to capability, registration time, timezone, or randomly in a distributed way.[7]
(4) *Task allocation phase*: Volunteers distribute tasks to their neighbors or appropriate volunteers by using distributed scheduling algorithms.
(5) *Task execution phase*: Each volunteer executes its task.
(6) *Task result return phase*: Each volunteer returns the result of its task to its parent volunteer.
(7) *Result certification phase* [8]: Some highly reliable volunteers or brokers check the correctness of the results returned from their child volunteers or the other volunteers managed by them.
(8) *Job result return phase*: The parent volunteers return the final results of the jobs to the client.

Computational overlay network (CON) is a logical set of volunteers for the execution of tasks [Milojicic et al. 2002; Barkai 2002; Steinmetz and Wehrle 2005; Subramanian and Goodman 2005]. The CON construction is similar to resource grouping except that CON construction is mainly performed by a volunteer or a broker in a distributed manner. In contrast, resource grouping is mainly performed by a server in a centralized way. In a distributed DG, scheduling is performed by each volunteer in a distributed way, depending on CON. In other words, volunteers distribute tasks to other volunteers differently according to the characteristics or topologies of CON (for example, tree, graph, or DHT(Distributed Hash Table)). A CON can be constructed on-the-fly or before-

[7]A broker can be responsible for the construction of CON (for example, tree according to registration time) [Neary et al. 1999].
[8]Most of the decentralized DG systems do not provide this phase.

scheduling. In the case of on-the-fly, CON construction and scheduling are performed at the same time [Chakravarti et al. 2005; Montresor et al. 2003; Zhong et al. 2003]. In the case of before-scheduling, CON construction is performed before scheduling [Neary et al. 1999]. Then, scheduling is performed on the basis of the structure of CON.

4.1.2 ***Platform.*** Desktop Grids are categorized into web-based (Java Applet-based) DG and middleware-based DG according to platform running on the volunteer's machine. In a *web-based DG*, clients write their parallel applications by using Java and post them as Applet on the Web. After that, participants join the web page with their browsers. At the moment, the Applet is downloaded automatically and runs on the participant's machine. Typical examples are Charlotte, Bayanihan, Javelin, and so on. In a *middleware-based DG*, participants need to install and run a specific middleware on their machine, which provides the services and functionalities for the execution of parallel applications. The middleware automatically fetches tasks from a server and executes them, when the CPU is idle. Typical examples are BOINC, XtremWeb, Entropia, Korea@Home, Alchemi and so on.

4.1.3 ***Scale.*** Desktop Grids are categorized into Internet-based DG and LAN-based DG according to scale. *Internet-based DG* is based on anonymous volunteers (see Table I). It should consider firewall, NAT(Network address translation), 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. It is also more controllable than Internet-based DG.

4.1.4 ***Resource Provider.*** Desktop Grids are categorized into volunteer DG and enterprise DG according to the properties of resource provider (see Table I). *Volunteer DG* is mainly based on voluntary participants. It mainly uses private resources owned and maintained by individuals. *Enterprise DG* is mainly based on non-voluntary participants within a corporation and a university. It mainly uses public resources owned and maintained by an institution. 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, XtremWeb, Bayanihan, Javelin, Korea@Home, and so on. Enterprise DG can be Entropia, Alchemi and Condor. It can be commercialized (for example, Entropia) or studied academically (for example, Condor, Alchemi).

4.2 Application Perspective

Desktop Grid systems should consider the following aspects on the application's perspective when designing and developing resource management and scheduling mechanisms (See Figure 10).

—**Type**: Is an application computation-intensive or data-intensive? In the case of data-intensive, a scheduler should consider data size, the location of data or replica, the cost of transfer, or replication policy, and so on [Venugopal et al. 2006]. In the case of computation-intensive, a scheduler focuses more on the

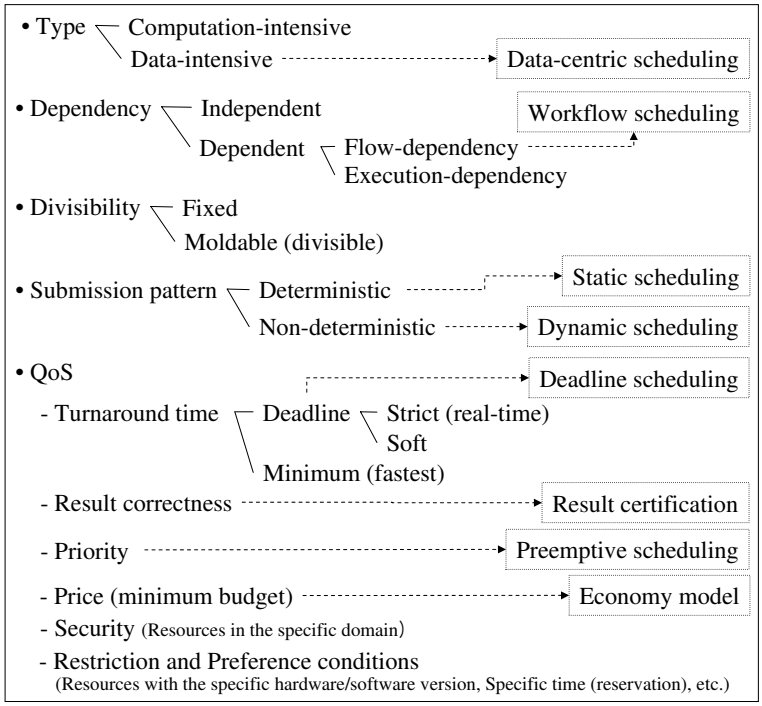

Fig. 10. A Taxonomy of Desktop Grids (Application Perspective)

resource's capability and availability.

—**Dependency**: Is there dependency between tasks? If there is no dependency between tasks, a scheduler attempts to allocate tasks to as many volunteers as possible according to the resource's availability, capability and execution properties, in order to complete as many tasks as possible [Anderson 2004; Neary et al. 1999; Zhou and Lo 2005; Kondo et al. 2004; Choi et al. 2006; Maheswaran et al. 1999; Braun et al. 2001]. In the case of dependent tasks, dependency can be categorized into flow and execution. The flow-dependency represents precedence and order between tasks (for example, workflow applications). On the other hand, the execution-dependency comes from interaction between tasks in the middle of execution (for example, MPI applications). In the case of flow-dependency, the relationship between tasks are mainly designed as a graph (for example, Directed Acyclic Graph (DAG)) [Yu and Buyya 2005; Cao et al. 2006]. The scheduler for DAG should consider the machine's capability, communication cost, data and task dependency, synchronization between tasks simultaneously, in order to minimize the overall execution time of the graph [Yu and Buyya 2005; Berman et al. 2003].

—**Divisibility**: Is a job flexibly divided into multiple tasks depending on the capability of resources during a scheduling procedure? In the case of a fixed job, a job is divided into the fixed size of tasks before scheduling. Then, they are distributed to volunteers. In the case of a divisible or moldable job, a sched-

uler decides the size (or amount) of a task during scheduling according to the resource's capability, deadline, etc. It focuses on how much of a task is assigned to a resource [Ali et al. 2005].

—**Submission pattern**: Does a client submit whole application and data to its scheduler before scheduling? Or, does it non-deterministically submit them during scheduling? In the deterministic case, if resources participate in the public execution irregularly, a scheduler should wait for them to start scheduling. In the non-deterministic case, if resources participate in public execution irregularly, it should wait for both tasks and resources.

—**QoS**: Some applications request QoS of Desktop Grid systems. A certain application needs to be finished before the deadline, or wants to do so rapidly. For example, a scheduler (that is, 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. Another application wants to guarantee result correctness. In the case, Desktop Grid systems should provide result certification or trust scheduling. A certain application with the highest priority wants to process more immediately and quickly than other applications. In this case, Desktop Grid systems allow a high-priority task to preempt a low-priority task running on a machine (that is, preemptive scheduling). In the non-preemptive scheduling, a machine is allowed to execute another task only after finishing a task. If an economy model is used, a client (that is, resource consumer) wants to buy resources minimizing cost/budget. A certain application does not want to be assigned to specific nodes or domain due to security, whereas another application wants to be assigned to specific nodes that satisfy the hardware/software version or quality required.

4.3 Resource Perspective

Desktop Grid systems should consider the following aspects on the resource's perspective when designing and developing resource management and scheduling mechanisms (See Figure 11).

—**Altruism**: Are resource owners willing to donate their individual resources for the public good? If resource owners are not altruistic [Ranganathan et al. 2004], that is, they are unwilling to share their resource although their whole CPUs are not used or even idle, then Desktop Grid systems should provide incentive mechanisms not only to participate in public execution, but also to encourage resource owners (volunteers) to contribute their resources more. Furthermore, they should provide incentive scheduling to give more benefits to eager volunteers in a scheduling procedure.

—**Dedication (or volatility)**: Are resources dedicated only to public execution without any interruption by private execution? Are resources allowed to freely leave in the middle of the public executions without any constraints? In Desktop Grid, resources are non-dedicated and volatile [Kondo et al. 2004; Brevik et al. 2004], so a public execution is suspended or stopped by a private execution. Therefore, Desktop Grid systems should deal with high-volatility and non-dedication. It is appropriate that a scheduler is opportunistic in the sense that a resource is not always available. Moreover, a scheduler can allocate tasks

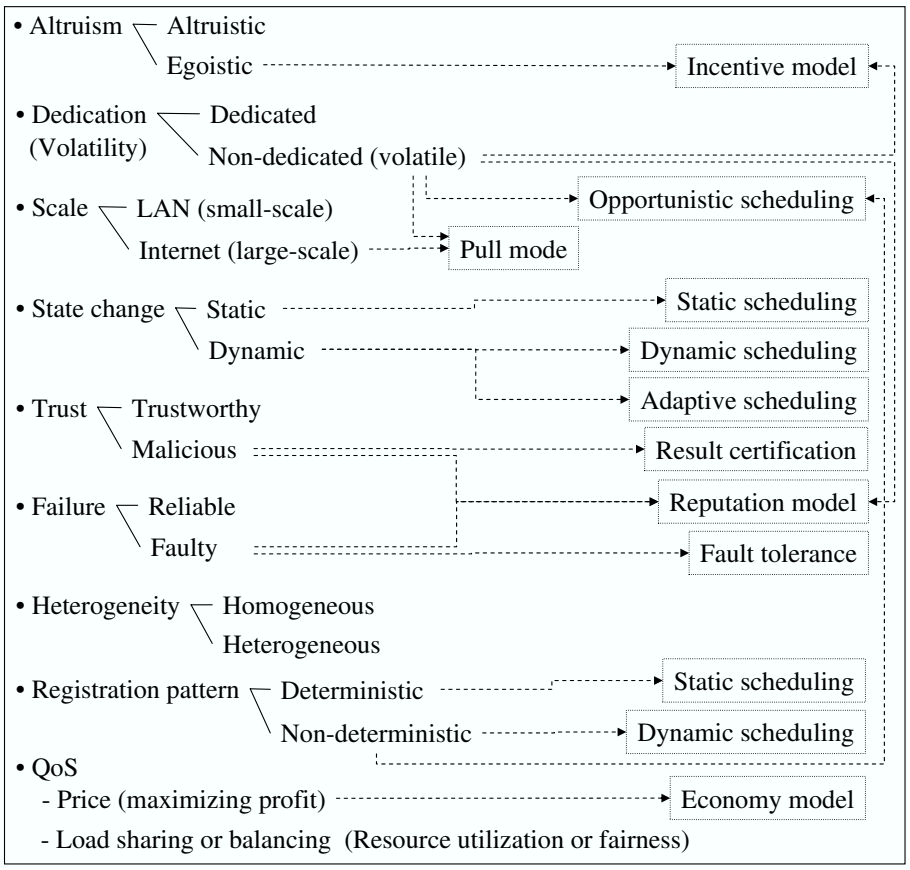

Fig. 11. A taxonomy of Desktop Grids (Resource Perspective)

on the basis of the volunteer's reputation (that is, characteristics and patterns of previous executions such as dedication and volatility) or be also coupled with incentive mechanisms in order to select eager resources or exclude selfish ones [Choi et al. 2006; Sonnek et al. 2007; Zhu et al. 2006]. To do this, Desktop Grid systems should provide reputation models to evaluate, score and rank volunteers.

—**Scale**: Are resources located in the scope of LAN or Internet? As shown in Table I, 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 the Internet, it is proper that scheduling events are initiated by a resource's request in the sense that some resources are behind NAT or firewall and they are not always available [BOINC ; Anderson 2004; Korea@Home]. In other words, resources pull a task from its scheduler (that is, pull mode). If resources are connected within LAN, Desktop Grid systems can manage them easily. Applications can be expanded, for example, workflow applications.

—**State change**: Is the execution 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, compared to Grid. In order to adapt to such a changing environment, Desktop Grid systems should monitor the state of volunteer and tasks and be able to cope with the state changes, for example, to change the scheduling policy dynamically and adaptively.

—**Trust**: Are resources trustworthy or malicious? If they are malicious, Desktop Grid systems need result certification in order to ensure the correctness of results. Particularly, a scheduler can be coupled with reputation in order to select trustworthy resources or exclude malicious ones [Choi et al. 2005; Sonnek et al. 2007; Zhu et al. 2006; Du et al. 2004].

—**Failure**: Are resources reliable or faulty? In Desktop Grid, volunteers are faulty. Therefore, Desktop Grid systems should provide fault tolerance for reliable execution. Particularly, a scheduler needs fault tolerant mechanisms (that is, checkpoint & restart, reassignment, replication, etc.). It also should deal with volatility because this leads to the failure of execution such as the delay and blocking of the executions of tasks and even partial or entire loss of the executions [Choi et al. 2004]. It can be coupled with reputation or incentive mechanism in order to select reliable resources or exclude faulty resources.

—**Heterogeneity**: Are resources heterogeneous or homogenous? 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, Desktop Grid has high execution heterogeneity. The execution heterogeneity makes scheduling more difficult and complex. It is necessary for a scheduler to be coupled with resource grouping, by which resources that have similar properties are grouped together, in the sense that a scheduler can apply scheduling, fault tolerance and result certification algorithms suitable for each group [Choi et al. 2006; Choi et al. 2006; 2005].

—**Registration pattern**: 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 [Choi et al. 2006; Casavant and Kuhl 1988; Rotithor 1994; Braun et al. 1998; Maheswaran et al. 1999]. In addition, opportunistic scheduling is needed because volunteers dynamically register.

—**QoS**: If economy model is used, a volunteer (that is, resource provider) wants to sell its resource maximizing its profit. 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 [Chow and Johnson 1997; Zhou 1988; Shivaratri et al. 1992]. Work stealing and load redistribution (that is, transferring tasks from heavily-loaded node to lightly-loaded node) improve resource utilization, fairness and performance.

4.4 Scheduler's Perspective

Desktop Grid systems should consider the following aspects on the scheduler's perspective when designing and developing resource management and scheduling mechanisms (See Figure 12 and 13).

—**Organization**: Scheduler organization is classified into three categories: centralized, distributed and hierarchical according to where and how scheduling decision is made [Krauter et al. 2002; Hamscher et al. 2000]. 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 decision is performed in a hierarchical way (for example, meta-scheduler (high-level scheduler) and local scheduler (low-level scheduler)). A high-lever scheduler allocates tasks to low-level schedulers, whereas a 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 by resources [Cappello et al. 2005; Chien et al. 2003; Tsaregorodtsev et al. 2004]. In other words, when a resource is idle or highly-loaded, it requests (or pulls) tasks to (or from) its server. Pull mode can be easily cooperated with opportunistic scheduling. In the **push mode**, a scheduler collects resource information, and then pushes tasks to resources [Tsaregorodtsev et al. 2004]. Generally, the pull mode is useful if resources are behind NAT or firewall, or if they are not dedicated [Anderson 2004; Chien et al. 2003; Choi et al. 2006; Tsaregorodtsev et al. 2004].

—**Policy**: Scheduling policy is used to match tasks with resources [Casavant and Kuhl 1988; Rotithor 1994; Braun et al. 1998; Ali et al. 2005; Maheswaran et al. 1999; Braun et al. 2001; Krauter et al. 2002; Yu and Buyya 2005; Yeo and Buyya 2006; Hamscher et al. 2000; Feitelson et al. 1997; Casanova et al. 2000]. It determines how to select appropriate tasks or resources. It is classified into three categories: simple, model-based and heuristics. 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 mathematics 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 tree topology, tasks are allocated from parent nodes to child nodes. In the economy model, scheduling decision is based on market (that is, price and budget) [Buyya et al. 2002; Yeo and Buyya 2006]. In the mathematics model, resources are selected in mathematics manners (such as Macov, Bayesian, genetic algorithm, game theory and machine learning techniques). In the **heuristics approach**, tasks or resources are selected by ranking, matching and exclusion methods on the basis of the resource's reputation or state. The reputation is related with the execution pattern and history (such as dedication, volatility, availability, credibility, etc.), whereas, the state represents the current state and capability of machines (such as hardware capability, performance, communication weight,

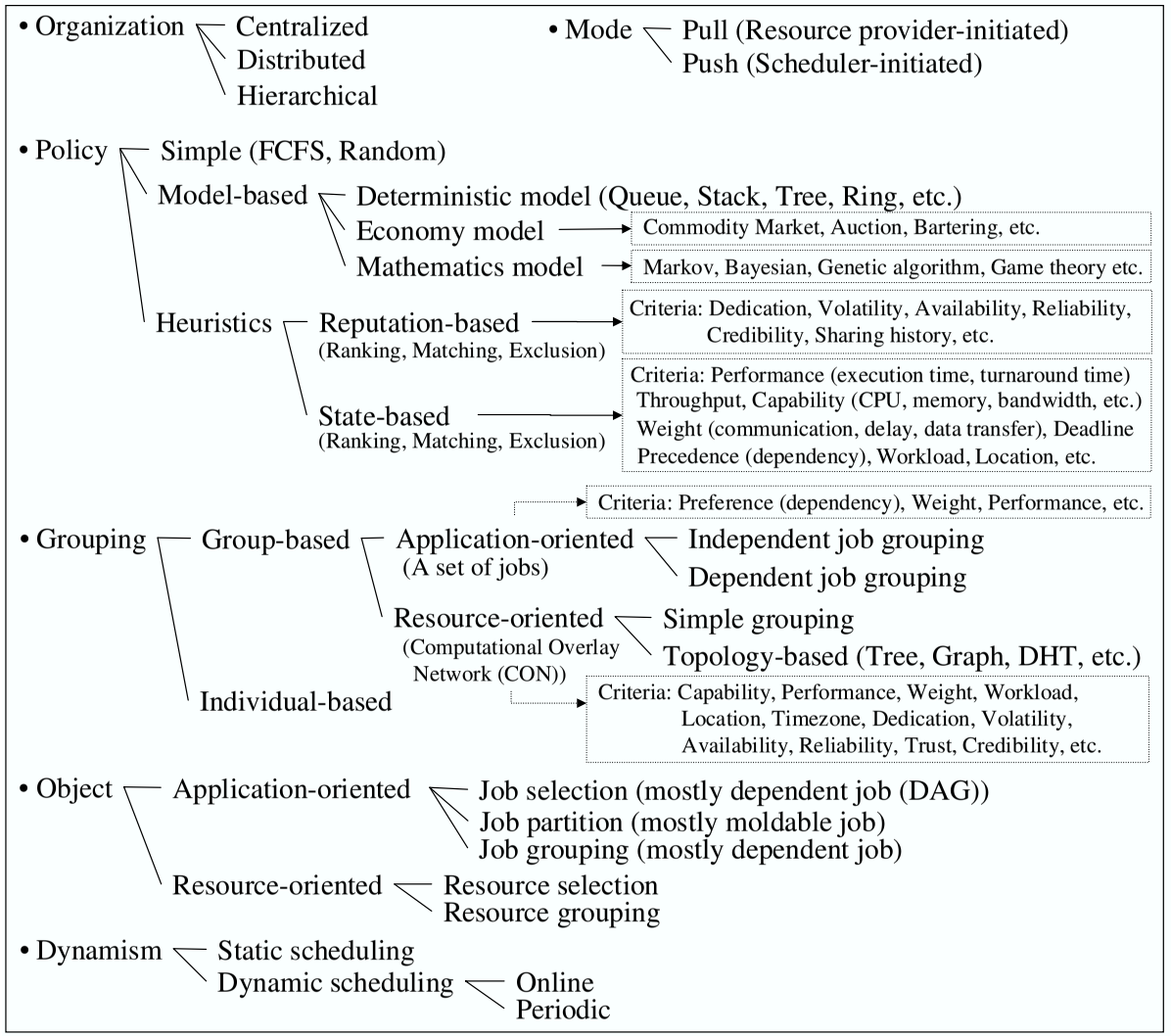

Fig. 12. A taxonomy of Desktop Grids (Scheduler Perspective)

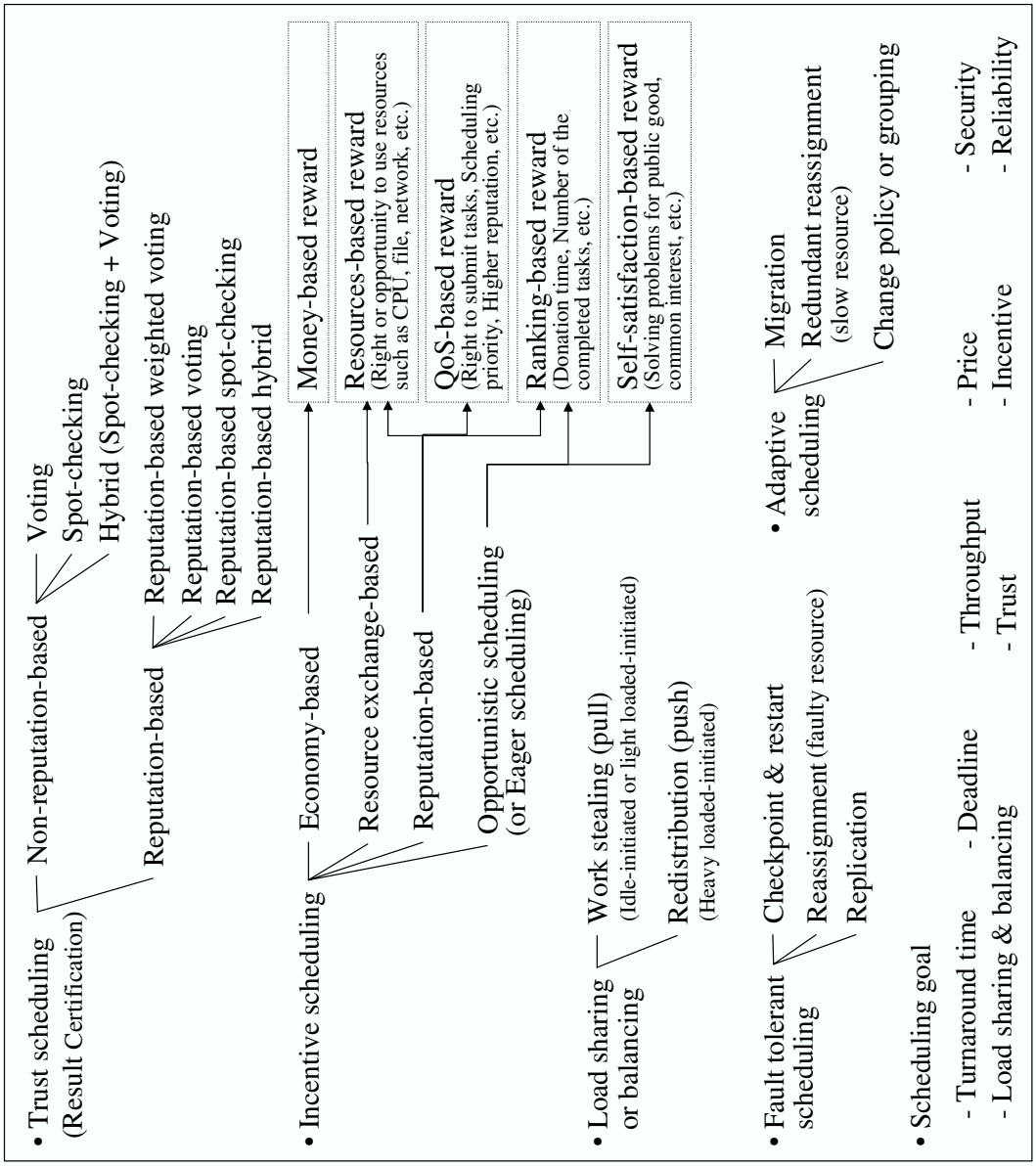

Fig. 13. A taxonomy of Desktop Grids (Scheduler Perspective)

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. [Maheswaran et al. 1999; Braun et al. 2001; Casanova et al. 2000]). 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.

—**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 [Yu and Buyya 2005; Cao et al. 2006]. Particularly, dependent tasks are grouped together on the basis of dependency or weight (communication or computation) in DAG, in order to improve performance or reduce communication cost [Yu and Buyya 2005; Cao et al. 2006]. For example, a set of tasks that uses the same data can be grouped together [Venugopal et al. 2006; Cao et al. 2006]. The **resource-oriented grouping approach** ensures that resources with similar properties are logically grouped together. The resource-oriented grouping approach constructs computational overlay networks (CONs). The characteristics and topology of CONs affect scheduling algorithms, resource management and information management. In addition, reliability, result correctness and performance depend on how CONs are 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 manner according to the target of scheduling. The **application-oriented approach** focuses on job selection, partition and grouping. Job selection and grouping focus on how to select resources or create a group according to the precedence and dependency of tasks. Job partition focuses on how much of a task is assigned to a resource. 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) [Yu and Buyya 2005; Cao et al. 2006], whereas an independent job is mainly related with resource-oriented approach (that is, to decide which resource is appropriate for a task) [Maheswaran et al. 1999; Braun et al. 2001; Cao et al. 2006].

—**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 [Casavant and Kuhl 1988; Rotithor 1994; Ekmecic et al. 1996; Ali et al. 2002; Maheswaran et al. 1999]. In the case of **static scheduling**, the prior information is assumed to be available [Casavant and Kuhl 1988; Rotithor 1994; Ekmecic et al. 1996; Ali et al. 2002; Maheswaran et al. 1999]. Static scheduling considers the entire tasks during decision making. In the case of **dynamic scheduling**, little a prior knowledge is available [Casavant and Kuhl

1988; Rotithor 1994; Ekmecic et al. 1996; Ali et al. 2002; Maheswaran et al. 1999]. 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 into account the environmental inputs 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 events occur [Maheswaran et al. 1999]. In the online approach, a scheduling event occurs as soon as a request arrives (for example, as soon as a resource arrives, or as soon as special conditions (that is, high or low workload threshold) happens). In contrast, in the case of the periodic approach, a scheduling event is performed only at the predefined conditions (for example, every predefined interval or time, or when the number of volunteers reaches a threshold); therefore some of the scheduling requests will be delayed until the predefined conditions are satisfied.

—**Trust scheduling (Result Certification)**: Trust scheduling for result certification aims to detect and tolerate the erroneous result in order to guarantee trusted execution. It is categorized into non-reputation-based and reputation-based trust scheduling. **Non-reputation-based trust scheduling** tolerates malicious resources or a variety of hardware and software malfunctions [Taufer et al. 2005; Sarmenta 2002; Choi et al. 2005; Renaud and Playez 2003] without using volunteer's reputation in a scheduling procedure. It is categorized into voting, spot-checking and hybrid. In the voting approach, the same task is distributed to different volunteers (that is, voting group) as many as the number of redundancy or until the predefined threshold is reached. If the results returned reach the predefined threshold (for example, the majority of volunteers generate the same result), they are considered as trustworthy. In the spot-checking approach, the special task whose result is already known is distributed to volunteers randomly selected. Then, the returned result is compared with the already-known result. If it is different, if it is not within specific error-tolerant range, or if the results returned do not reach the predefined threshold, the volunteers are regarded as malicious and the results returned are discarded. Voting approach is apparently more costly than spot-checking, because it requires a redundancy of at least two per task. Spot-checking and voting can be combined together. In a **reputation-based trust scheduling**, result certification is coupled with the reputation of volunteers. In this case, the more a volunteer produces a correct result, the higher its reputation (especially, credibility) becomes. In a reputation-based weighted voting, some volunteers' votes carry more weight than others according to their reputation. In a reputation-based voting, a scheduler calculates the redundancy of voting according to the volunteer's reputation, or excludes badly-reputed volunteers in a scheduling procedure. In a reputation-based spot-checking, a scheduler calculates the rate of spot-checking according to the volunteer's reputation, or selects badly-reputed volunteers for the test, rather than highly-reputed volunteers.

—**Incentive scheduling**: Incentive mechanisms [Obreiter and Nimis 2003; Zhu et al. 2006; Ranganathan et al. 2004; Andrade et al. 2007] aim to encourage re-

sources' owners to donate their resources eagerly, reliably and trustworthily. It gives rewards (such as, money, resources, ranking, etc.) to volunteers for their donation. Incentive scheduling also has the same goals, but it is more related with resource selection and management [Zhu et al. 2006; Ranganathan et al. 2004; Andrade et al. 2007]. It tries to give incentives to eager, reliable and trustworthy volunteers in a scheduling procedure. It also attempts to inflict punishment (for example, penalty, exclusion from scheduling, low reputation and ranking, etc.) on selfish, faulty, untrustworthy volunteers. Incentive scheduling is categorized into economy-based, resource exchange-based, reputation-based, opportunistic scheduling (or eager scheduling). In an economy-based incentive scheduling [9], a scheduler tries to allocate tasks to eager, reliable and trustworthy volunteers in order to give them more money. A resource exchange-based incentive scheduling is mainly applied to distributed DG systems. A volunteer accepts tasks submitted by volunteers that have already donated their resources to it. In the reputation-based incentive scheduling, a scheduler distributes tasks according to the volunteer's reputation. For example, only volunteers that have high or same reputation are allowed to use other resources [Ranganathan et al. 2004], or to submit tasks. In addition, a scheduler tries to give much higher reputation and ranking to volunteers that donate their resources eagerly, reliably and trustworthily. In an opportunistic scheduling (or eager scheduling), volunteers that request a task are first served in order to give an opportunity of participation. In general, they are satisfied with the donation itself because they are resource providers who have common interest or want to solve problems for public good.

—**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.

—**Fault tolerant scheduling**: Fault tolerant scheduling tolerates failure and volatility. It involves selecting more reliable resources according to availability, volatility, or credibility in order to avoid failures as much as possible, and performing reassignment or replication in the presence of failures or volatility [Taufer et al. 2005; Sarmenta 2002; Choi et al. 2006; Choi et al. 2006; 2005; Choi et al. 2004; Abawajy 2004; Anglano et al. 2006; Lee et al. 2005]. It is classified into checkpoint & restart, reassignment and replication. In the **checkpoint & restart approach**, if a scheduler detects the failures of resource, it restarts the failed task at another resource from the checkpoint. In the **reassignment approach**, if a scheduler detects the failures of resource, 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 can mask the failure.

[9]Economy model has incentives in itself because it gives money to resource providers. Economy model focuses more on maximization or minimization of the profit of resource providers or consumers, whereas economy-based incentive scheduling focuses on the selection of eager, reliable and trustworthy volunteers in order to give them more money as an incentive.

—**Adaptive scheduling**: Adaptive scheduling takes environmental stimuli into account to adapt to dynamically changing environment [Casavant and Kuhl 1988; Rotithor 1994; Shivaratri et al. 1992]. The environmental changes lead 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 changes. 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 [Maheswaran et al. 1999].

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

5. MAPPING OF TAXONOMY TO STATE-OF-THE-ART SYSTEMS

A mapping of the taxonomy to Desktop Grid systems or projects is illustrated in this section. Existing Desktop Grid systems, projects, and papers are surveyed (see Table II and III) : Alchemi [Luther et al. 2005], Bayanihan [Sarmenta and Hirano 1999; Sarmenta 2002; 2001], BOINC [BOINC ; Anderson 2004; Taufer et al. 2005; Anderson et al. 2005], Cluster Computing On the Fly(CCOF) [Zhou and Lo 2004; Lo et al. 2004; Zhou and Lo 2005; Zhao and Lo 2005], Charlotte [Baratloo et al. 1999], Condor [Thain et al. 2003; 2005; Tannenbaum et al. 2003], Computer Power Market (CPM) [Buyya and Vazhkudai 2001; Ping et al. 2001], Entropia [Chien et al. 2003; Chien et al. 2003], Javelin [Neary et al. 1999; Neary et al. 2000; Neary and Cappello 2005], Korea@Home [Korea@Home ; Choi et al. 2007; Choi et al. 2006; Choi et al. 2006; 2005; Byun et al. 2007; Choi et al. 2004], Messor [Babaoglu et al. 2002; Montresor et al. 2003], Organic Grid [Chakravarti et al. 2005; 2006], Paradropper [Zhong et al. 2003; Dou et al. 2003], POPCORN [Nisan et al. 1998], WebCom [Morrison et al. 2001; 2002], XtremWeb [Fedak et al. 2001; Cappello et al. 2005] and Kondo et al. [Kondo et al. 2002; Kondo et al. 2004; Kondo et al. 2004; Kondo 2005; Kondo et al. 2006].

Table IV shows the survey of existing Desktop Grid systems according to the taxonomy of Desktop Grid shown in Figure 7. Table V shows the survey of existing Desktop Grid systems according to the application's perspective shown in Figure 10. Tables VI and VII show the survey of existing Desktop Grid systems according to the resource's perspective shown in Figure 11. Tables VIII, IX, X, XI show the survey of existing Desktop Grid systems focusing on scheduling according to the scheduler's perspective shown in Figures 12 and 13.

Table II. Desktop Grid Projects around the World

System	Developer	Remarks	Availability
Alchemi	University Of Melbourne	-.NET based Enterprise Desktop Grid -http://www.alchemi.net/	Open source
Bayanihan	MIT	- Volunteer computing system (Java or .NET) -Credibility-based eager scheduling -http://bayanihancomputing.net/	No
BOINC	University Of California, Berkeley	- Volunteer computing and Desktop Grid computing (C++) -SETI@Home,Predictor@Home,Folding@Home, Climatepredication.net,LHC@Home,Einstein@Home, etc. -http://boinc.berkeley.edu/	Open source
CCOF	University Of Oregon	-Peer-based Desktop Grid computing -Wave scheduling, Trust-based scheduling -http://ccof.cs.uoregon.edu/	No
Charlotte	New York University	-Metacomputing on the Web (Java) -Eager scheduling -http://www.cs.nyu.edu/milan/charlotte/	Open source
Condor	University Of Wisconsin Madison	-High throughput computing (C, C++) -http://www.cs.wisc.edu/condor/	Open binary
CPM	University Of Melbourne	-Market-based Grid computing over the Internet -http://www.computepower.com/	No
Entropia	Entropia Inc.	-Enterprise Desktop Grid -VolunteerDG(ENTROPIA2000),EnterpriseDG(DCGRID™) -http://www.entropia.com	No

Table III. Desktop Grid Projects around the World (Continued)

System	Developer	Remarks	Availability
Javelin	University Of California, Aanta Barbara	-Internet-based Globally Distributed computing (Java) -Tree based advanced eager scheduling -http://www.cs.ucsb.edu/projects/javelin/	No
Korea@Home	Korea Institute of Science and Technology Information (KISTI)	-PC-based distributed computing platform (C++) -Group-based scheduling and Markov-based scheduling -http://www.koreaathome.org/eng/	No
Messor (Anthill)	University Of Bologna	-Agent-based peer-to-peer systems (Java) -Agent-based load balancing -http://www.cs.unibo.it/projects/anthill/	Open source
Organic Grid	Louisiana State University	-A fully decentralized Desktop Grid -Self-organizing scheduling -http://csc.lsu.edu/~gb/OrganicGrid/	No
Paradropper	Changsha Institute of Technology	-General purpose global computing environment (Java) -Small world graph-based scheduling	No
Popcorn	Hebrew University Of Jerusalem	-Global distributed computation over the Internet (Java) - Auction-based economy scheduling -http://www.cs.huji.ac.il/~popcorn/	Open source
WebCom	University College Cork	-Volunteer-based metacomputer -http://www.cuc.ucc.ie/research	No
XtremWeb	Paris XI University	-Global computing experimental platform (Java) -http://www.lri.fr/~fedak/XtremWeb/	Open source

Table IV. Survey of Desktop Grid systems (System perspective)

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

5.1 Alchemi

Alchemi [Luther et al. 2005] is a .NET-based Desktop Grid computing framework that aggregates the computing power of networked desktop computers. It provides a .NET API and tools to develop .NET-based Grid applications. Alchemi is being used in applications such as large scale document processing, CSIRO Australia's hydrology application, and Microsoft Excel spreadsheet processing.

Alchemi consists of a manager (server in this article) and executors (volunteer in this article). A manager is in charge of the management of executors and scheduling which is based on priority and First Come First Served (FCFS). It contains a cross-platform manager, which is a web services interface that exposes the functionality that is capable of translating a non-canonical job submitted by a user into a form that can be acceptable by a manager. An executor executes threads (tasks in this article). Alchemi can be configured as two modes, that is, enterprise DG and Volunteer DG. In the case of enterprise DG, a manager can explicitly instruct the executor to execute threads (that is, push mode). In the case of volunteer DG, an executor requests threads from the manager. If it is disconnected, the threads are rescheduled.

Table V. Survey of Desktop Grid systems (Application perspective)

System	App. Type	Dependency	Divisibility	Submission pattern	QoS
Alchemi	Compute-intensive	Independent, Dependent (workflow)	Fixed	Deterministic	Not specified
Bayanihan	Compute-intensive	-Independent -Dependent (BSP)	Fixed	Deterministic	Result correctness
BOINC	Compute-intensive, Data-intensive	Independent	Fixed	Deterministic	-Priority -Result correctness
CCOF	Compute-Intensive	Independent	Fixed	Non-deterministic	Result correctness
Charlotte	Compute-intensive	Independent	Fixed	Deterministic	Not specified
Condor	Compute-intensive, Data-intensive	Independent, Dependent	Fixed	Deterministic, Non-deterministic	-Priority -Restriction or Preference conditions
CPM	Compute-intensive	Independent	Fixed	Deterministic	-Price -Deadline
Entropia	Compute-Intensive	Independent	Fixed	Deterministic	Not specified
Javelin	Compute-intensive	Independent	Fixed	Non-deterministic	Not specified
Korea@home	Compute-Intensive	Independent	Fixed	Deterministic	Result correctness
Messor	Compute-Intensive	Independent	Fixed	Non-deterministic	Not specified
Organic Grid	Compute-Intensive	Independent	Fixed	Non-deterministic	Fast turnaround time
Paradropper	Compute-intensive	Independent	Fixed	Non-deterministic	Not specified
Popcorn	Compute-intensive	Independent	Fixed	Deterministic	Price
WebCom	Compute-intensive	Dependent (Condensed graph)	Fixed	Deterministic	Not specified
XtremWeb	Compute-Intensive	Independent	Fixed	Deterministic	Not specified
Kondo et al.	Compute-intensive	Independent	Fixed	Deterministic, Non-deterministic	Fast turnaround time

Table VI. Survey of Desktop Grid systems (Resource perspective)

System	Altruism	Dedication (Volatility)	Scale	State change	Trust	Failure	Hetero-geneity	Registration pattern	QoS
Alchemi	Altruistic	-Volatile, Non-dedicated, -Dedicated (possible)	-Internet -LAN	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
Bayanihan	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Untrust-worthy	Faulty	Hetero-geneous	Non-deterministic	Not specified
BOINC	Altruistic, Egoistic (Credit)	-Volatile, Non-dedicated, -Dedicated (possible)	Internet	Dynamic	Untrust-worthy	Faulty	Hetero-geneous	Non-deterministic	Not specified
CCOF	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Untrust-worthy	Not specified	Hetero-geneous	Non-deterministic	Not specified
Charlotte	Altruistic	Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
Condor	Altruistic	-Volatile & Non-dedicated, -Dedicated	-Internet -LAN	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
CPM	Egoistic (Economy -based)	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Price
Entropia	Altruistic	-Volatile & Non-dedicated, -Dedicated	-Internet -LAN	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified

Table VII. Survey of Desktop Grid systems (Resource perspective)

System	Altruism	Dedication (Volatility)	Scale	State change	Trust	Failure	Hetero-geneity	Registration pattern	QoS
Javelin	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
Korea@home	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Untrust-worthy	Faulty	Hetero-geneous	Non-deterministic	Not specified
Messor	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Not specified	Hetero-geneous	Non-deterministic	Load balancing
Organic Grid	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
Paradropper	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Not specified	Hetero-geneous	Non-deterministic	Not specified
Popcorn	Egoistic (Economy -based)	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Price
WebCom	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified
XtremWeb	Altruistic	Non-dedicated, Volatile	Internet	Dynamic	Untrust-worthy	Faulty	Hetero-geneous	Non-deterministic	Not specified
Kondo et al.	Altruistic	-Volatile & Non-dedicated, -Dedicated (possible)	-Internet -LAN	Dynamic	Not specified	Faulty	Hetero-geneous	Non-deterministic	Not specified

Table VIII. Survey of Desktop Grid systems (Scheduler perspective)

System	Organization	Mode	Policy	Grouping	Object	Dynamism
Alchemi	Centralized	Push Pull	FCFS	Individual-based	Resource oriented: resource selection	Dynamic *Online
Bayanihan	Centralized	Pull	-FCFS (eager scheduling) -Heuristics: Reputation-based *Exclusion (credibility)	Individual-based	Resource oriented: resource selection	Dynamic *Online
BOINC	Centralized	Pull	-FCFS	Individual-based	Resource oriented: resource selection	Dynamic *Online *Periodic
CCOF	Distributed	Push	-Heuristics: State-based *Matching (timezone) -Heuristics: Reputation-based *Ranking, exclusion (trust)	Resource-oriented: CON (Timezone based DHT)	Resource oriented: resource grouping, resource selection	Dynamic *Online *Periodic
Charlotte	Centralized	Pull	FCFS (eager scheduling)	Individual-based	Resource oriented: resource selection	Dynamic *Online
Condor	Centralized	Push Pull	Heuristics: State-based *Matching, ranking (capability, location, workload)	Individual-based	Resource oriented: resource selection	Dynamic *Online *Periodic
CPM	Distributed	Pull	Economy model: *Commodity market-oriented	Individual-based	Resource oriented: resource selection	Dynamic *Online
Entropia	Centralized	Pull	-Deterministic model: *Queue (job selection) -Heuristics-based: *Matching (capability)	Resource-oriented : Simple grouping (capability)	Resource oriented: resource selection, resource grouping	Dynamic *Online
Javelin	Distributed	Pull	-Random -Deterministic model: *Tree-based eager scheduling	Resource-oriented : CON (tree)	Resource oriented: resource selection, resource grouping	Dynamic *Online

Table IX. Survey of Desktop Grid systems (Scheduler perspective)

System	Organization	Mode	Policy	Grouping	Object	Dynamism
Korea@Home	Centralized	Pull	-FCFS -Heuristics: Reputation-based *Matching, ranking, exclusion (availability, credibility, dedication, volatility) -Mathematics model (Markov)	Individual-based or Resource-oriented: Simple grouping (availability, credibility, volatility)	Resource oriented: resource grouping, resource selection	Dynamic *Online *Periodic
Messor	Distributed	Push Pull	-Random -Heuristics: State-based *Matching (workload)	Resource-oriented: CON (Workload-based random graph)	Resource oriented: resource selection	Dynamic *Online
Organic Grid	Distributed	Pull	Deterministic model *Tree	Resource-oriented: CON (Performance-based tree)	Resource oriented: resource grouping, resource selection	Dynamic *Online
Paradropper	Distributed	Push	Heuristics: State-based *Ranking (workload)	Resource-oriented: CON (small world graph)	Resource oriented: resource grouping, resource selection	Dynamic *Online
Popcorn	Centralized	Pull	Economy-model *Auction-based	Individual-based	Resource oriented: resource selection	Dynamic *Online
WebCom	Hierarchical	Pull	-Deterministic model: *Queue (round-robin) -Heuristics: State-based *Matching (capability-network, latency, performance)	Resource-oriented: CON (tree)-latency	Resource oriented: resource selection	Dynamic *Online
XtremWeb	Centralized	Pull	-FCFS	Individual-based	Resource oriented: resource selection	Dynamic *Online
Kondo et al.	Centralized	Pull Push	-Heuristics: State-based *Ranking, exclusion (capability-CPU); Exclusion (performance); Ranking (throughput)	Individual-based	Resource oriented: resource selection	Dynamic *Online *Periodic

Table X. Survey of Desktop Grid systems (Scheduler perspective)

System	Trust scheduling	Incentive scheduling	Load sharing & balancing	Fault tolerance	Adaptive scheduling	Scheduling Goal
Alchemi	Not supported	Not supported	Not supported	Reassignment	Reassignment	-Reliability
Bayanihan	-Result certification *Voting *Spot-checking *Hybrid (voting+spot-checking) -Reputation-based: *Trust threshold (Credibility-based eager scheduling)	Eager scheduling *Self-satisfaction-based reward	Not supported	Reassignment	Reassignment	-Trust (sabotage tolerance) -Reliability
BOINC	-Result certification *Voting (homogeneous redundancy)	Opportunistic scheduling *Ranking-based reward *Self-satisfaction-based reward	Not supported	-Checkpoint /restart -Replication	Reassignment	-Throughput -Trust -Incentive -Reliability
CCOF	-Result certification *Voting *Spot-checking (quiz & replication in collision)	Reputation-based *QoS-based reward (scheduling priority, Higher reputation)	Not supported	Not supported	Migration (timezone)	-Turnaround time -Trust -Incentive
Charlotte	Not supported	Eager scheduling *Self-satisfaction-based reward	Not supported	Reassignment	Reassignment	-Throughput -Reliability
Condor	Not supported	Opportunistic scheduling *Self-satisfaction-based reward	Not supported	Checkpoint /restart	Migration	-Throughput -Reliability
CPM	Not supported	Economy-based scheduling *Money-based reward	Not supported	Reassignment	Not supported	-Price -Deadline
Entropia	Not supported	Not supported	Not supported	Reassignment	Reassignment	-Throughput
Javelin	Not supported	Not supported	Work stealing (Pull)	Reassignment	Reassignment	-Reliability -Load balance

Table XI. Survey of Desktop Grid systems (Scheduler perspective)

System	Trust scheduling	Incentive	Load balancing	Fault tolerance	Adaptive scheduling	Scheduling Goal
Korea@Home	-Reputation based: *Voting *Spot-checking	Reputation based *QoS-based reward (Scheduling priority, Higher reputation)	Not supported	-Reassignment -Replication (Group-based)	-Reassignment -Change policy or grouping	-Turnaround time -Throughput -Trust -Incentive -Reliability
Messor	Not supported	Not supported	-Work stealing (pull) -Load redistribution (push)	Not supported	Change policy *Push : overloaded *Pull : light loaded	Load balance
Organic Grid	Not supported	Not supported	Not supported	Reassignment	Change topology (performance-based)	Turnaround time
Paradropper	Not supported	Not supported	Load redistribution (push)	Not supported	Not supported	Load balance
Popcorn	-Result Certification *Voting * Spot-checking	Economy-based scheduling * Money-based reward	Not supported	Reassignment	Not supported	Price
WebCom	Not supported	Not supported	Not supported	Reassignment	Change topology (Tree: load, latency)	Reliability
XtremWeb	Not supported	Not supported	Not supported	Reassignment	Reassignment	Reliability
Kondo et al.	Not supported	Not supported	Not supported	Replication (Duplication)	Reassignment	-Turnaround time -Throughput -Deadline -Reliability

5.2 Bayanihan

Bayanihan [Sarmenta and Hirano 1999; Sarmenta 2002; 2001] is a web-based volunteer computing system using Java. Bayanihan system consists of client (volunteer in this article) and server. A client executes Java applet on a web browser (that is, web-based), or Java application on a middleware (that is, middleware-based). It has a worker engine that executes computation or a watcher engine that shows 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.

A server (that is, work manager) 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 eagerly volunteers work, the more tasks are executed. Additionally, the credibility-based fault tolerance mechanism was studied to tolerate erroneous results from malicious volunteers [Sarmenta 2002; 2001]. In the majority voting approach, the same task is performed by different volunteers as much as the number of redundancy. In a spot-checking approach, the special task whose result is already known is performed by the volunteers randomly selected. If a volunteer returns an erroneous result, it is regarded as malicious one. In the credibility-enhanced eager scheduling, a task is continuously allocated to volunteers until its credibility threshold is satisfied. When the desired credibility threshold is reached, the result is accepted as a final. A volunteer's credibility is calculated as follows. The more a volunteer passes the spot-checking, the higher its credibility becomes. The more volunteers within voting group agree on the same result, the higher its credibility becomes. The volunteers that produce erroneous results can be blacklisted. Moreover, Bayanihan also has fault-tolerant and adaptive scheduling algorithms. 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 [Sarmenta 2001], which provides message-passing and remote memory primitives.

5.3 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [BOINC ; Anderson 2004; Taufer et al. 2005; Anderson et al. 2005] 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 [BOINC ; Anderson 2004; Taufer et al. 2005; Anderson et al. 2005].

BOINC consists of server and client (volunteer in this article). 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) executes projects' applications. It can participate in several projects and specify preferences for the projects. BOINC is mainly based on voluntary participants connected through the Internet[10].

The BOINC server is responsible for scheduling. Clients (that is, volunteers) first send requests to its task server (that is, pull mode). Then, it allocates new tasks to them. Additionally, BOINC supports locality scheduling. Moreover, clients performs local scheduling on its computer[11], which decides which task to run among multiple projects, when to ask a server for more works, which project to ask, and how much work to ask for [BOINC].

BOINC is used for applications in physics, molecular biology, medicine, chemistry, astronomy, climate, etc. The applications are mainly compute-intensive and independent. The BOINC scheduler distributes a task to clients only if they are likely to complete the task by its deadline. If the deadline passes or if a task fails, the server marks it as time-out and then redistributes it to other volunteers. With the local scheduling, a client may preempt applications either by suspending them or by instructing them to quit if it participates in multiple projects [BOINC]. In other words, one task can be preempted by another task. BOINC also provides the checkpoint API for applications, that is, when to checkpoint and when a checkpoint has been done.

BOINC supports redundant computing to detect and tolerate erroneous results. It provides homogeneous redundancy for the numerical applications that may produce different results depending on the machine architecture, operating systems, compiler, and compiler flags [Taufer et al. 2005]. In this case, the redundant tasks are dispatched to numerically identical computers.

5.4 CCOF

CCOF(Cluster Computing On the Fly) [Zhou and Lo 2004; Lo et al. 2004; Zhou and Lo 2005; Zhao and Lo 2005] 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.

Hosts (volunteers in this article) join a community based overlay network (CAN-based DHT overlay) to donate their idle cycles [Zhou and Lo 2004; Lo et al. 2004]. Clients discover these resources from this overlay network, 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[12]. If the host fails to find a new host, the original client reschedules the job. This scheduling is called wave scheduling in that tasks ride a wave of idle cycles (day or night timezone). Wave scheduling supports deadline-driven tasks. Scheduling is close to a distributed model in the sense that there is no server and migration is performed by each host, although a client initially schedules the tasks.

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

[11]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.

[12]Migration can be applicable to a push model

The result verification schemes (that is, Quiz and replication) were studied on the assumption that there is collusion among malicious hosts [Zhao and Lo 2005]. The Quiz method inserts a quiz task (its result is known to a client) into a normal task (that is, it is similar to spot-checking). In addition, the trust-based scheduling was proposed; it uses the reputation system to select trusted hosts [Zhao and Lo 2005]. The reputation system evaluates trust values of hosts according to the result of result verification. The malicious hosts can be blacklisted.

5.5 Charlotte

Charlotte [Baratloo et al. 1999] is a Java-based infrastructure for metacomputing on the Web. It consists of manager (server in this article) and worker (volunteer in this article). 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, a worker asks for and executes each task. After that, if it finishes its task, it contacts the 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, it reassigns one of the unfinished tasks to the worker. This redundant assignment of a task to multiple workers eventually tolerates the slow workers and the 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.

5.6 Condor

Condor [Thain et al. 2003; 2005; Tannenbaum et al. 2003] 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 [Thain et al. 2005]. Condor is comprised of a central manager (server in this article) and other resources (volunteer in this article). 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 boundary of an organization. Condor can be configured to run jobs only when the keyboard and CPU are idle. While 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 [Thain et al. 2005; Tannenbaum et al. 2003]. It also provides a matchmaker for matching tasks with available resources. Condor performs matchmaking as follows. Agent (that is, client) and resources advertise their ClassAds to its matchmaker. The matchmaker investigates the ClassAds, and selects job and resource pairs that satisfy each other's constraints and preferences. Finally, they establish a contract, and then cooperate to execute tasks.

ClassAd describes the special attributes: *Requirements* and *Rank* [Tannenbaum et al. 2003]. *Requirements* attribute indicates a constraint, and *Rank* attribute

measures the desirability of a match. A 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 [Tannenbaum et al. 2003]. 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.

5.7 CPM

CPM(Compute Power Market) [Buyya and Vazhkudai 2001; Ping et al. 2001] is a market-based middleware system for Grid computing on low-end personal computing devices connected to the 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 across Internet-wide volunteer resources.

CPM consists of a market, a resource consumer (client in this article), and a resource provider (volunteer in this article). A market is a mediator between consumer and provider. It maintains information about providers and consumers. A resource consumer buys computing power from a market. It downloads a market resource broker 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 with resource providers, and distributes tasks to them. The application and data files are fetched when the task is ready to run at a resource provider. A resource provider sells computing power through the market. It 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) with help of a market resource broker. Scheduling is performed by 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 the resource's cost with providers and there is no special server that is responsible for scheduling.

5.8 Entropia

Entropia [Chien et al. 2003; Chien et al. 2003] is a middleware system for commercial Desktop Grid. Entropia provides two solutions: enterprise Desktop Grid (Entropia DCGrid) and Internet Grid (Entropia 2000). The applications are mainly compute-intensive and independent.

Entropia consists of server and client (volunteer in this article). 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 (that is, volunteers) within a pool have similar capabilities such as disk space or operating system type. The App server decomposes a job into subjobs and assigns them to clients. The task server performs

scheduling in the pull mode.

Enterprise Desktop Grid consists of three layers: job management, subjob management, and resource management layers. 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, the 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 the 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.

5.9 Javelin

Javelin [Neary et al. 1999; Neary et al. 2000; Neary and Cappello 2005] 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 article). A client registers its tasks to a broker. A host provides computing resources. A broker coordinates the supply and demand for computing resources. When a host contacts a broker, the broker adds it to a logical tree structure. A broker maintains and reorganizes the tree of hosts (that is, tree-based CON).

A client registers with a broker. If a host requests tasks to the broker, the broker informs it of client ID and application information. Then the host executes tasks. At this time, work stealing and advanced eager scheduling are performed [Neary et al. 2000; Neary and Cappello 2005]. With work stealing, when a host runs out of work, it requests tasks of other hosts in two ways: deterministic or probabilistic approaches. In a deterministic approach, a host asks its children or its parents for tasks on the basis of tree structure. In a probabilistic approach, a host randomly selects the target from the list of other hosts that it currently keeps. With the advanced eager scheduling, the client selects the next task marked undone and redistributes 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 the 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 by a host, and eager scheduling is performed by 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.

5.10 Korea@Home

The Korea@Home project attempts to harness the massive computing power of the great numbers of desktop computers over the Internet. Korea@Home system consists of server and agent (volunteer in this article). A server is responsible for scheduling and information management about jobs and agents. Korea@Home has applications: Genome alignment, New drug candidate discovery, rainfall forecast, climate predication, optical analysis of TFT LDC. Most applications are mainly compute-intensive and independent.

Korea@Home basically uses FCFS scheduling. If a volunteer is idle, it requests a task of its server(that is, pull mode). Additionally, a group-based adaptive scheduling and a Markov-based scheduling mechanism were studied. Choi et al. [Choi et al. 2006; Choi et al. 2006; 2005; Choi et al. 2004] proposed centralized scheduling mechanisms in Desktop Grid. They proposed a group-based adaptive scheduling mechanism, which evaluates volunteers according to their reputations such as availability, dedication and credibility, then forms volunteer groups. Then, it applies scheduling, replication, result certification, and fault tolerant algorithms to each group while ranking or excluding volunteers. Particularly, it calculates the number of replication, the rate of result certification on the basis of volunteer group, therefore it reduces overhead, and completes more tasks. It can also tolerate volunteer's volatility and non-dedication (which is called volunteer autonomy failures) as well as crash and link failures during execution. Byun et al. [Byun et al. 2007] proposed a probabilistic scheduling using the Markov model. It provides the stochastic model of volunteer's availability using the Markov, and then distributes tasks to volunteers selected by scheduling algorithms such as optimist, pessimist, and realist methods.

5.11 Messor

Messor[13] [Babaoglu et al. 2002; Montresor et al. 2003] 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 the *SerarchMax* state. It selects the next nest randomly or according to workload. When it finds an overloaded node, it records the identifier of this nest and changes its state into the *SearchMin* state. From now on, it wanders through the network, looking for a light-loaded nest. When it finds a light-loaded nest, it requests the local job manager on the nest to fetch jobs from this overloaded nest, and then changes its state back to the *SearchMax* state. Ants continuously perform the processes within its time-to-live. When a task is completed, the result is sent back to the original nest. Like this, Messor constructs

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

workload-based random graph on the fly and achieves load balancing.

5.12 Organic Grid

Organic Grid [Chakravarti et al. 2005; 2006] provides a self-organizing and distributed approach to the organization of the computation. Hosts (volunteer in this article) 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. At the moment, 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 parent 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.

5.13 Paradropper

Paradropper [Zhong et al. 2003; Dou et al. 2003] is a global computing system that supports self-organizing overlay network by using peer-to-peer technologies. Peers (volunteers in this article) are organized into an overlay network by using small world characteristics. A new peer sends 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 it 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, it results in load balancing. In addition, powerful peers have executed more tasks than weaker peers.

5.14 POPCORN

POPCORN [Nisan et al. 1998] is a Java-based infrastructure for globally distributed computation over the Internet. POPCORN provides a market-based mechanism for the trade of computational resources. POPCORN system consists of market, seller (volunteer in this article), and buyer (client in this article). 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, transferring tasks and results between them, and handling all payments and accounts. It is a well-known meeting place or matchmaker for buyers and sellers. It uses several auction models (that is, a repeated Vickrey auction, a sealed bid double-auction, and repeated Clearinghouse double auction) for matching buyers and sellers [Nisan et al. 1998].

POPCORN deals with the failure and verification. If a task fails, it simply resends the tasks to a different host. POPCORN simply uses replication, spot-checking, self-testing, etc.[14]

5.15 WebCom

WebCom [Morrison et al. 2001; 2002] is a web-based distributed computation platform using Java. It consists of master (server in this article) and client (volunteer in this article). A master maintains a set of clients and is responsible for scheduling. A client receives tasks as the form of Java applet, executes them 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 by 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 load. If a task fails, it is rescheduled to another client.

5.16 XtremWeb

XtremWeb [Fedak et al. 2001; Cappello et al. 2005] 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 the Internet.

XtremWeb is composed of client, server (or coordinator) and worker (volunteer in this article). 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

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

initiated by a worker. A worker contacts its server to get tasks. In other words, 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.

5.17 Kondo et al.

Kondo et al. [Kondo et al. 2002; Kondo et al. 2004; Kondo et al. 2004; Kondo 2005; Kondo et al. 2006] proposed centralized scheduling mechanisms in Desktop Grid. They studied resource selection approaches for short-lived application: resource prioritization, resource exclusion, and task replication [Kondo et al. 2004]. 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. [Kondo et al. 2002] proposed timeout mechanism. If a server does not receive result from a host within timeout, it redistributes the task to another host. In addition, they proposed a scheduling mechanism by using a buffer in order to complete applications within deadline [Kondo et al. 2006].

6. DISCUSSION

On the basis of the taxonomy and mapping, the current states and characteristics of Desktop Grid systems are discussed. Then the challenging issues are extracted and the future directions for Desktop Grids are presented.

6.1 Analysis of Desktop Grid Systems

6.1.1 *System perspective.* Both centralized DG and distributed DG have been developed. Particularly, centralized DG systems are publicly used by users and scientists. Centralized DG can implement and develop resource management, scheduling algorithms, result certification, and reputation/incentive mechanisms more easily than distributed DG, even though it has scalability problems. With regard to platform, initial DG systems are most web-based, while current DG systems have adopted middleware-based, which can easily apply to a screen saver mode. In terms of scale and resource provider, most of the DG systems are based on volunteers through the Internet (that is, volunteer DG). Recently, Desktop Grid systems try to support enterprise DG as well as volunteer DG, because enterprise DG gives many benefits, for examples, more reliable and trusted environment and easier manageability.

6.1.2 *Application perspective.* The application type of Desktop Grid systems is mostly computation-intensive. Only a few Desktop Grid systems such as BOINC and Condor (that is, LAN-based DG or enterprise DG) support data-intensive applications, because it is expensive to transfer data across the Internet. In terms of dependency, most of the Desktop Grid systems support only independent applications, because it is very difficult to control communication between volunteers on

the Internet because of non-dedication, volatility and failures. With regard to divisibility, Desktop Grid systems focus on scheduling of the fixed size of tasks, because it is difficult to calculate how much of a task is assigned to heterogeneous resources, respectively. In terms of submission pattern, centralized DG systems are mainly deterministic, and distributed DG systems are non-deterministic. This is why any volunteers in a distributed DG can submit tasks in a distributed way. With regard to QoS, some Desktop Grid systems provides resource management, result certification, and scheduling algorithms that support the application's demands such as result correctness, fast turnaround time, deadline and price. Especially, result correctness is implemented in centralized DG systems[15].

6.1.3 *Resource perspective.* Most of the Desktop Grid systems assume that volunteers are altruistic. Therefore, they do not provide any incentive mechanisms. Some Desktop Grid systems such as BOINC, Korea@Home, CPM, and Popcorn use the ranking or economy model to encourage donation. Volunteers are volatile and non-dedicated in Desktop Grid systems. LAN-based and enterprise DG such as Alchemi, Entropia, and Condor can set volunteers dedicated. Volunteers are connected through the Internet in most of the Desktop Grid systems. A few Desktop Grid systems such as Alchemi, Entropia and Condor are based on volunteers within LAN. In all the Desktop Grid systems, volunteers are dynamically changing. A few Desktop Grid systems such as Bayanihan, BOINC and Korea@Home deal with the trust of volunteers. They provide result certification to detect and tolerate erroneous results. Most of the Desktop Grid systems consider faulty volunteers. They provide fault tolerant scheduling (for example, reassignment or replication). In all the Desktop Grid systems, volunteers are heterogeneous and non deterministic. A few Desktop Grid systems such as Messor, CPM and Popcorn gives load sharing and balancing and supports economy model in order to satisfy QoS.

6.1.4 *Scheduler perspective.* Most centralized DG systems[16] provides centralized scheduling, distributed DG systems provides distributed scheduling. Most of the Desktop Grid systems use the pull mode, because volunteers are volatile and non-dedicated, or are connected behind firewall and NAT. Enterprise DG systems (Alchemi and Condor) and some distributed DG systems (CCOF, Messor and Paradropper) provide the push mode. In terms of policy, most of the centralized DG systems use FCFS (or eager scheduling), some of them (Condor and Entropia) use state-based heuristics. CPM and Popcorn provide economy model. Most of the Desktop Grid systems do not use reputation-based heuristics, but just provide state-based heuristics. With regard to grouping, most of the centralized DG systems do not provide grouping methods, whereas distributed DG systems provide them. Most of the distributed DG systems construct CON according to workload and performance, or randomly. Most of the Desktop Grid systems focus on resource selection and grouping, not application selection, because most of the tasks are independent and volunteers are heterogeneous and dynamic. In Desktop Grid systems, scheduling is dynamic because volunteers dynamically come and go. Most of the Desktop Grid systems do not provide trust scheduling. Especially, dis-

[15]In CCOF (distributed DG), result verification is studied in a centralized way.
[16]WebCom provides hierarchical scheduling on the basis of tree structure.

tributed DG systems do not provide result certification, because it is very difficult to implement it in a distributed way. Most of the Desktop Grid systems do not provide incentive mechanisms and do not couple incentive with scheduling. Even the Desktop Grid systems that provide incentive scheduling mostly use a basic incentive (self satisfaction-based reward), a few systems use reputation-based and economy-based scheduling. Most of the distributed DG systems do not provide incentive scheduling. In terms of load sharing and balancing, some of the distributed DG systems provide work stealing and load redistribution. Most of Desktop Grid systems provide fault tolerant scheduling, that is, reassignment and replication. In terms of adaptive scheduling, most of centralized DG systems provide reassignment for slow volunteers, and most of distributed DG systems change topology or policy in response to the conditions of volunteer and execution. With regard to scheduling goal, most of Desktop Grid schedulers aim for turnaround time, throughput, and reliability. Most of the Desktop Grid schedulers do not provide trust and are not couple with incentive.

6.2 Challenging Issues of Desktop Grid Systems

Desktop Grid has challenging problems such as volatility, dynamicity, 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. Moreover, the public executions get temporarily suspended by a private execution because volunteers are not totally dedicated only to public executions. Accordingly, volunteers have various volunteering time (that is, the time of donation). They also have the various occurrence rate and form of volatility, which directly affect the execution of tasks. The volatility and non-dedication make Desktop Grid systems difficult to provide reliable computation as well as good performance.

—***Dynamicity***: 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, workload, availability, volatility, reputation, trust, failure, etc.) is continuously changing over time during the public execution. Desktop Grid systems should adapt to such a dynamic environment for performance and reliable execution.

—***Lack of trust***: In Desktop Grid, anonymous nodes can participate as a resource provider. Some malicious volunteers tamper with the computation and then return the corrupted results. Desktop Grid systems should deal with malicious volunteers in order to guarantee trusted execution and the correctness of results.

—***Failure***: In Desktop Grid, volunteers are mainly connected through the 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. Desktop Grid systems should tolerate the failures and volatility for reliable execution.

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

—***Scalability***: Centralized DG 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. Therefore, Centralized DG should be scalable for better performance and management. On the contrary, distributed DG provides scalability because scheduling decision is made at each volunteer. However, it is difficult to apply trusted scheduling and incentive. It also has poor performance as compared with centralized DG 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. Therefore, it is difficult to collects volunteers. In order to encourage resource providers to participate in public execution and donate their resources eagerly for a long time, Desktop Grid systems should provide incentive mechanisms. Moreover, they should couple reputation and incentive with scheduling to give more benefits and rewards.

6.3 Future Directions for Desktop Grids

Desktop Grid has largely two research areas: applications and systems, as shown in Figure 14. The applications part studies how to implement and develop applications suitable for Desktop Grid. On the other hand, the systems part focuses on how to organize and provide various functionalities and components to support applications and manage resources. The systems part has sub-research topics such as scheduling, resource management, communication and security. These topics can be combined together towards better performance and reliability. For example, scheduling can be coupled with resource grouping, reputation, or incentive. Trust can be coupled with reputation.

To overcome the above challenging issues, Desktop Grid systems should more deal with the following considerations. The relationship between challenging issues and future directions is illustrated in Figure 15.

—***Computational overlay network (resource grouping)***: In Desktop Grid, volunteers should be grouped together according to their properties (such as capability, performance, workload, reputation, 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 [Choi et al. 2006; Choi et al. 2006; 2005]. Thus, it is very important how to make the CON depending on the volunteer's properties, because scheduling, resource management, and information

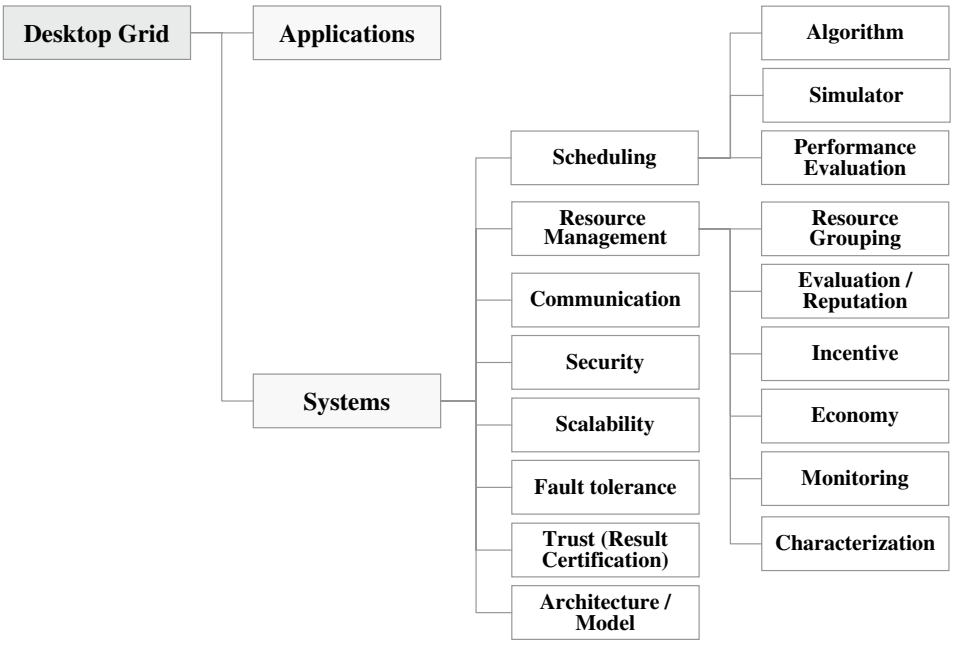

Fig. 14. Research Areas of Desktop Grid

management are performed on the basis of characteristics or topologies of the CON. However, existing centralized DG systems [Anderson 2004; Fedak et al. 2001; Sarmenta and Hirano 1999; Nisan et al. 1998] do not provide how to construct the CON on the basis of reputation such as volunteering time, volatility, trust and credibility. They simply construct a CON depending on the resource's basic properties (for example, disk space, OS type, etc.). On the other hand, existing decentralized Desktop Grid systems [Neary et al. 1999; Zhou and Lo 2004; Chakravarti et al. 2005; Montresor et al. 2003; Zhong et al. 2003] provide how to construct a CON depending on registration time, timezone, performance, or workload, but they do not consider volatility, volunteering time, credibility and trust, which directly affect reliability, completion time, and result correctness. Moreover, resource grouping is not tightly coupled with scheduling (especially, result certification, replication and reassignment). As a result, uncoupling between resource grouping and scheduling causes a lot of overhead of replication and result certification and degrades performance. Desktop Grid systems should provide more delicate construction methods of CON and couple resource grouping with scheduling. Additionally, future systems need to aim at supporting negotiation based resource grouping and allocation according to client's requirements.

—***Reputation & Incentive Mechanisms***: 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 and donation patterns. Therefore, Desktop Grid systems

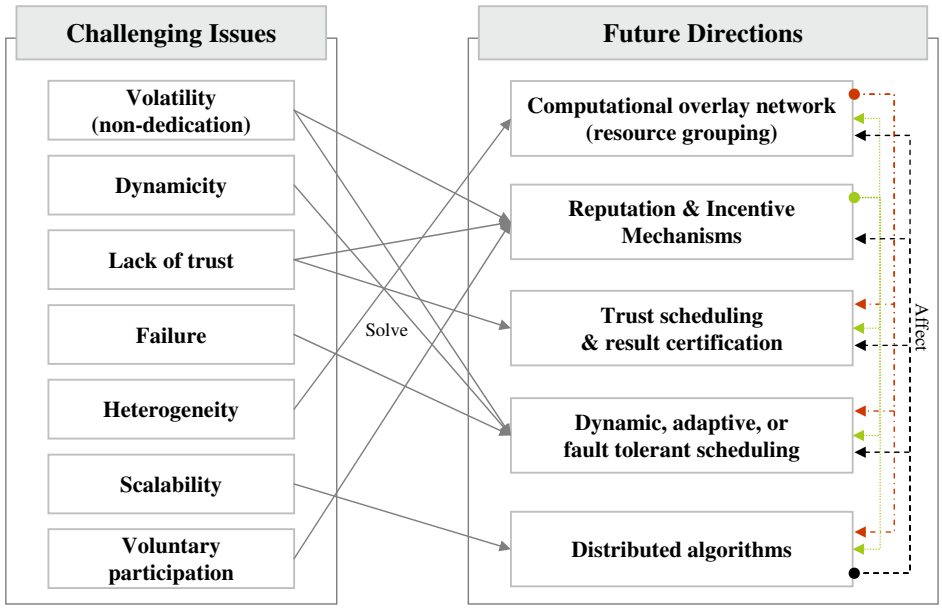

Fig. 15. The relationship between challenging issues and future directions

consider reputation and incentive mechanisms in order to score and rank the volunteers, and then reward or punish them according to the assessment. Moreover, a reputation system should be coupled with scheduling. A reputation-based scheduling enables selection of highly qualified resources, so that it can improve the reliability, trust and performance. An incentive scheduling tries to give more rewards and benefit to eager, reliable and trustworthy volunteers, and punish volatile, selfish, or malicious volunteers (for example, exclusion). Thus, it encourages volunteers to donate their resources eagerly and reliably. Desktop Grid systems should consider reputation-based and incentive scheduling.

—***Trust Scheduling & 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 [Taufer et al. 2005]. Therefore, Desktop Grid systems need to detect and tolerate the erroneous result in order to guarantee a reliable execution in such a distrusted environment. To this end, Desktop Grid systems exploit result certification mechanisms such as majority voting and spot-checking [Taufer et al. 2005; Sarmenta 2002; Zhao and Lo 2005; Choi et al. 2005; Renaud and Playez 2003]. 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, not considering volunteer's reputation. As a result, there are high overhead, performance degradation, and scalability

problems. Desktop Grid systems should provide new trust scheduling mechanisms for result certification, coupling resource's reputation such as volatility, volunteering time, and credibility with result certification.

—***Dynamic, adaptive, or fault tolerant scheduling***: Desktop Grid is based on desktop computers at the edge of the 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, reputation, etc.) are changing over time. Desktop Grid systems should adapt to dynamically changing environment. In other words, Desktop Grid systems should be able to obtain dynamically changing state and then consider them as environmental inputs or stimuli in a scheduling procedure when making decisions. Particularly, a scheduler should be able to deal with volatility and failures that occur frequently in a dynamic environment, in order to provide reliability.

—***Distributed algorithms***: Distributed DG [Neary et al. 2000; Lo et al. 2004; Chakravarti et al. 2005; Montresor et al. 2003; Dou et al. 2003] 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 reputation such as volatility, volunteering time, credibility and trust, which directly affect reliability, completion time and result correctness. Moreover, they do not provide incentive or result certification, fault tolerance mechanisms, which is necessary in Desktop Grid. Distributed DG systems should provide incentive or result certification, fault tolerance mechanisms, which are coupled with volunteer's reputation.

7. RELATED STUDIES

Even though Desktop Grid has been studied, there is no general survey and taxonomy 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. Moreover, although resource management and scheduling in Desktop Grid is different from Grid, there is no survey or taxonomy of Desktop Grid. As a result, it is difficult to design and develop new resource management and scheduling mechanisms.

7.1 Taxonomy and Survey of Grid and Desktop Grid

There are several taxonomy and survey of Grid. Baker et al. [Baker et al. 2002] attempted to present the state-of-the-art of Grid computing and survey emerging Grid computing projects. Krauter et al. [Krauter et al. 2002] proposed the taxonomy of Grid focusing on resource management. Venugopal et al. [Venugopal et al. 2006] proposed the taxonomy of Data Grids according to organization, data transport, data replication and scheduling. Foster et al. [Foster and Iamnitchi 2003] compared P2P and Grid computing.

Sarmenta [Sarmenta 2001] classified volunteer computing into application-based and web-based (java-based). Chien et al. [Chien et al. 2003] classified Desktop Grid into Internet Grid and Enterprise Grid. Cappello [Cappello 2007] and Chu et al. [Chu et al. 2007] classified Desktop Grids into the first (that is, application

specific systems), the second (that is, multi-application systems), and the third generation (that is, modular and service-oriented systems). However, they do not provide a mapping of taxonomy to the existing Desktop Grid systems. Moreover, the taxonomy proposed in this article is more delicate and expanded.

7.2 Taxonomy and Survey of Scheduling and Resource Management

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

Casavant et al. [Casavant and Kuhl 1988] 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 [Rotithor 1994] proposed the taxonomy of dynamic scheduling in distributed computing systems according to state estimation and decision making.

Braun et al. [Braun et al. 1998] 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. [Ali et al. 2005] characterized resource allocation heuristics for heterogeneous computing systems according to workload, platform, and mapping strategy[17]. Ekmecic et al. [Ekmecic et al. 1996] modified the taxonomy proposed by Casavant et al. [Casavant and Kuhl 1988]. Maheswaran et al. [Maheswaran et al. 1999] 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. [Braun et al. 2001] 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. [Krauter et al. 2002] 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. [Yu and Buyya 2005] proposed the taxonomy of scientific workflow scheduling for Grid computing. The taxonomy is defined in four aspects: architecture, decision making, planning scheme, and scheduling strategies. Yeo et al. [Yeo and Buyya 2006] 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. [Venugopal et al. 2006] proposed the taxonomy of scheduling for Data Grid according to application model, scope, data replication, utility function and locality. Hamscher et al. [Hamscher et al. 2000] proposed centralized, hierarchical, and decentralized scheduling architectures for Grid.

[17]The three categories of our taxonomy proposed in this article (that is, application, resource, and scheduler's perspective) is similar to the three categories of the taxonomy proposed by Barun et al [Braun et al. 1998] and Ali et al. [Ali et al. 2005]. 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.

Although several studies have investigated taxonomy of heterogeneous computing systems and various aspects of Grid systems, but no works have focused on Desktop Grid systems. Hence, the taxonomy proposed in this article makes contributions towards enhancing the understanding of Desktop Grid and their development. Particularly, the proposed 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.

8. CONCLUSION

In this article, a new taxonomy of Desktop Grids was proposed in order to characterize and categorize Desktop Grid systems. In addition, a mapping of the taxonomy to the Desktop Grid systems was presented. Detailed investigation have made on various Desktop Grid technology and development during the last one decade. Through the taxonomy and survey, the challenging issues of Desktop Grids have discovered such as volatility, dynamicity, lack of trust, failure, heterogeneity, scalability and voluntary participation. To overcome challenging issues, more research on Desktop Grids will be needed such as 1) computational overlay network (resource grouping), 2) reputation or incentive-oriented scheduling and resource management, 3) trust scheduling for result certification, 4) dynamic, adaptive, or fault tolerant scheduling, and 5) distributed algorithms. Moreover, further research on QoS service and support for different applications model will be undertaken. The taxonomy and survey will help understand the architecture, model, and characteristics of Desktop Grid.

Acknowledgment

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2007-357-D00207) and the Department of Computer Science and Software Engineering, University of Melbourne.

REFERENCES

Abawajy, J. H. 2004. Fault-tolerant scheduling policy for grid computing systems. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), the 5th International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-04)*. IEEE, Santa Fe, USA, 238b.

Abbas, A. 2003. *Grid Computing: A Practical Guide to Technology and Applications*. Charles River Media, Hingham, USA.

Ali, S., Braun, T., Siegel, H., Maciejewski, A., N.Beck, Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., and Yao, B. 2005. Characterizing resource allocation heuristics for heterogeneous computing systems. *Advances in Computers: Parallel, Distributed and Pervasive Computing 63*, 93–129.

Ali, S., Braun, T. D., Siegel, H. J., and Maciejewski, A. A. 2002. Heterogeneous computing. In *Encyclopedia of Distributed Computing*. J. Urbana and P. Dasgupta, Eds. Kluwer Academic Publishers, Norwell, USA.

Anderson, D. P. 2004. Boinc: A system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*. IEEE CS Press, Pittsburgh, USA, 4–10.

Anderson, D. P., Korpela, E., and Walton, R. 2005. High-performance task distribution for volunteer computing. In *Proceedings of the First IEEE International Conference on e-Science and Grid Technologies (e-Science2005)*. IEEE CS Press, Melbourne, Australia, 196–203.

Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. 2007. Automatic grid assembly by promoting collaboration in peer-to-peer grids. *Journal of Parallel and Distributed Computing 67,* 8 (Aug.), 957–966.

Anglano, C., Brevik, J., Canonico, M., Nurmi, D., and Wolski, R. 2006. Fault-aware scheduling for bag-of-tasks applications on desktop grids. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*. IEEE, Barcelona, Spain, 56–63.

Babaoglu, O., Meling, H., and Montresor, A. 2002. Anthill: a framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*. IEEE CS Press, Vienna, Austria, 15–22.

Baker, M., Buyya, R., and Laforenza, D. 2002. Grids and grid technologies for wide-area distributed computing. *Software: Practice and Experience 32,* 15 (Dec.), 1437–1466.

Baratloo, A., Karaul, M., Kedem, Z. M., and Wijckoff, P. 1999. Charlotte: Metacomputing on the web. *Future Generation Computer Systems, Special Issue on Metacomputing 15,* 5-6 (Oct.), 559–570.

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

Berman, F., Fox, G. C., and Hey, A. J. G. 2003. *Grid Computing : Making the Global Infrastructure a Reality*. Wiley, Chichester, West Susses.

Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., and Zagorodnov, D. 2003. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems 14,* 4 (Apr.), 369–382.

BOINC. *http://boinc.berkeley.edu/*.

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

Braun, T. D., Siegely, H. J., Becky, N., Boloniz, L., Maheswarany, M., Reuthery, A. I., Robertsony, J. P., Theysy, M. D., and Yaoy, B. 1998. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS 1998)*. IEEE CS Press, West Lafayette, USA, 330–335.

Brevik, J., Nurmi, D., and Wolski, R. 2004. Automatic methods or predicting machine availability in desktop grid and peer-to-peer systems. In *Proceedings of the 4th IEEE International symposium on Cluster Computing and the Grid (CCGRID 2004)*. IEEE, Chicago, USA, 190–199.

Buyya, R., Abramson, D., Giddy, J., and Stockinger, H. 2002. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience 14,* 13-15 (Nov.), 1507–1542.

Buyya, R. and Vazhkudai, S. 2001. Compute power market: towards a market-oriented grid. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'01)*. IEEE CS Press, Brisbane, Australia, 574–581.

Byun, E. J., Choi, S. J., Baik, M. S., Gil, J. M., Park, C. Y., and Hwang, C. S. 2007. Mjsa: Markov job scheduler based on availability in desktop grid computing environment. *Future Generation Computer Systems 23,* 4 (May), 616–622.

Cao, J., Chan, A. T., Sun, Y., Das, S. K., and Guo, M. 2006. A taxonomy of application scheduling tools for high performance cluster computing. *Cluster Computing 9,* 3 (July), 355–371.

Cappello, F. 2007. 3rd generation desktop grids. In *Proceedings of the 1st XtremWeb Users Group Workshop (XW'07)*. Hammamet, Tunisia.

Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Neri, V., and Lodygensky, O. 2005. Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems 21,* 3 (Mar.), 417–437.

Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. 2000. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*. IEEE CS Press, Cancun, Mexico, 349–363.

Casavant, T. L. and Kuhl, J. G. 1988. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering 14,* 2 (Mar.), 141–154.

Chakravarti, A., Baumgartner, G., and Lauria, M. 2005. The organic grid: Self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics 35,* 3 (May), 1–12.

Chakravarti, A., Baumgartner, G., and Lauria, M. 2006. The organic grid: Self-organizing computational biology on desktop grids. In *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*. A. Y. ZOMAYA Eds. Wiley-Interscience, Hoboken, USA, 431–450.

Chien, A., Calder, B., Elbert, S., and Bhatia, K. 2003. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing 63,* 5 (May), 597–610.

Chien, A. A., Marlin, S., , and Elbert, S. T. 2003. Resource management in the entropia system. In *Grid Resource Management: Sate of the Art and Future Trends*. J. NABRZYSKI, J. M. SCHOPF and J. WEGLARZ Eds. Kluwer Academic, 431–450.

Choi, S. J., Baik, M. S., Gil, J. M., Jung, S. Y., and Hwang, C. S. 2006. Adaptive group scheduling mechanim using mobile agents in peer-to-peer grid computing environment. *Applied Intelligence, Special Issue on Agent-based Grid Computing 25,* 2 (Oct.), 199–221.

Choi, S. J., Baik, M. S., Gil, J. M., Park, C. Y., Jung, S. Y., and Hwang, C. S. 2005. Dynamic scheduling mechanism for result certification in peer to peer grid computing. In *Proceedings of International Conference on Grid and Cooperative Computing (GCC 2005)*. LNCS 3795, Springer-Verlag, Beijing, China, 811–824.

Choi, S. J., Baik, M. S., Gil, J. M., Park, C. Y., Jung, S. Y., and Hwang, C. S. 2006. Group-based dynamic computational replication mechanism in peer to peer grid computing. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID2006), Sixth Workshop on Global and Peer-to-Peer Computing (GP2P)*. IEEE CS Press, Singapore, 7.

Choi, S. J., Baik, M. S., Hwang, C. S., Gil, J. M., and Yu, H. C. 2004. Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment. In *Proceedings of the 3th IEEE International Symposium on Network Computing and Applications, Workshop on Adaptive Grid Computing (NCA-AGC2004)*. IEEE CS Press, Cambridge, USA, 476–483.

Choi, S. J., Kim, H. S., Byun, E. J., Baik, M. S., Kim, S. S., Park, C. Y., and Hwang, C. S. 2007. Characterizing and classifying desktop grid. In *Proceedings of the Sixth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2007), Workshop on Global and Peer to Peer Computing (GP2P)*. IEEE CS Press, Rio de Janeiro, Brazil, 743–748.

Chow, R. and Johnson, T. 1997. *Distributed Operating Systems & Algorithms*. Addison-Wesley.

Chu, X., Nadiminti, K., Jin, C., Venugopal, S., and Buyya, R. 2007. Aneka: Next-generation enterprise grid platform for e-science and e-business applications. In *Proceedings of the 3rd International International Conference on e-Science and Grid Computing (e-Science 2007)*. IEEE CS Press, Bangalore, India, 151–159.

Distributed.net. *http://distributed.net*.

Dou, W., Jia, Y., Wang, H. M., Song, W. Q., and Zou, P. 2003. A p2p approach for global computing. In *Proceedings of International Parallel and Distributed Processing Symposium 2003 (IPDPS 2003)*. IEEE CS Press, Nice, France, 6–11.

Du, W., Jia, J., Mangal, M., and Murugesan, M. 2004. Uncheatable grid computing. In

Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004). IEEE CS Press, Tokyo, Japan, 4–11.

Ekmecic, I., Tartalja, I., and Milutinovic, V. 1996. A survey of heterogeneous computing: concepts and systems. *Proceedings of the IEEE 84,* 8 (Aug.), 1127–1144.

Fedak, G., Germain, C., Neri, V., and Cappello, F. 2001. Xtremweb: A generic global computing system. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001): Workshop on Global Computing on Personal Devices*. IEEE CS Press, Brisbane, Australia, 582–587.

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. 1997. Theory and practice in parallel job scheduling. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1997)*. LNCS 1291, Springer-Verlag, Geneva, Switzerland, 1–34.

Foster, I. and Iamnitchi, A. 2003. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*. LNCS 2735, Springer-Verlag, Berkeley, USA, 24–31.

Foster, I. and Kesselman, C. 2004. *The Grid 2: Blueprint for a new Computing Infrastructure*. Morgan Kaufmann, San Francisco, USA.

GIMPS. *http://www.mersenne.org/*.

Hamscher, V., Schwiegelshohn, U., Streit, A., and Yahyapour, R. 2000. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (Grid 2000)*. LNCS 1971, Springer-Verlag, Bangalore, India, 191–202.

Jacob, B., Ferreira, L., Bieberstein, N., Gilzean, C., Girard, J. Y., Strachowski, R., and Yu, S. S. 2003. *Enabling Applications for Gird Computing with Globus*. IBM Redbooks.

Kondo, D. 2005. Scheduling task parallel applications for rapid turnaround on desktop grids. Ph.D. thesis, University of California, San Diego, San Diego, USA.

Kondo, D., Casanova, H., Wing, E., and Berman, F. 2002. Models and scheduling mechanisms for global computing applications. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*. IEEE CS Press, Fort Lauderdale, USA, 79–86.

Kondo, D., Chien, A. A., and Casanova, H. 2004. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC2004)*.

Kondo, D., Kindarji, B., Fedak, G., and Cappello, F. 2006. Towards soft real-time applications on enterprise desktop grids. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID2006)*. IEEE CS Press, Singapore, 65–72.

Kondo, D., Taufer, M., Karanicolas, J., Brooks, C. L., Casanova, H., and Chien, A. 2004. Characterizing and evaluating desktop grids: An empirical study. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*.

Korea@Home. *http://www.koreaathome.org/eng/*.

Krauter, K., Buyya, R., and Maheswaran, M. 2002. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience 32,* 2 (Feb.), 135–164.

Lee, H. M., Chung, K. S., Chin, S. H., Lee, J. H., Lee, D. W., Park, S. B., and Yu, H. C. 2005. A resource management and fault tolerance services in grid computing. *Journal of Parallel and Distributed Computing 65,* 11 (Nov.), 1305–1317.

Li, M. and Baker, M. 2004. *The Grid: Core Technologies*. John Wiley & Sons Ltd, Chichester, West Susses.

Lo, V., Zhou, D., Zappala, D., Liu, Y., and Zhao, S. 2004. Cluster computingon the fly: P2p scheduling of idle cycles in the internet. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*. LNCS 3279, Springer-Verlag, San Diego, USA, 227–236.

Luther, A., Buyya, R., Ranjan, R., and Venugopal, S. 2005. Alchemi: A .net-based enterprise grid computing system. In *Proceedings of the 6th International conference on Internet Computing (ICOMP'05)*. Las Vegas, USA, 269–278.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., and Freund, R. F. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*. IEEE CS Press, San Juan, USA, 30–44.

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., J. Pruyne, B. R., Rollins, S., and Xu, Z. 2002. Peer-to-peer computing. *HP Laboratories Palo Alto HPL-2002-57*.

Montresor, A., Meling, H., and Babaoglu, O. 2003. Messor: Load-balancing through a swarm of autonomous agents. In *Proceedings of International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*. LNCS 2530, Springer-Verlag, Bologna, Italy, 125–137.

Morrison, J. P., Kennedy, J. J., and Power, D. A. 2001. Webcom: A web based volunteer computer. *Journal of Supercomputing 18,* 1 (Jan.), 47–61.

Morrison, J. P., Kennedy, J. J., and Power, D. A. 2002. Load balancing and fault tolerance in a condensed graphs based metacomputer. *The Journal of Internet Technologies, Special Issue on Web based Programming 3,* 4 (Dec.), 221–234.

Neary, M. O., Brydon, S. P., Kmiec, P., Rollins, S., and Cappello, P. 2000. Javelin++: Scalability issues in global computing. *Concurrency: Parctice and Experience 12,* 8 (Aug.), 727–753.

Neary, M. O. and Cappello, P. 2005. Advanced eager scheduling for java-based adaptive parallel computing. *Concurrency and Computation: Practice and Experience 17,* 7-8 (June), 797–819.

Neary, M. O., Christiansen, B. O., Cappello, P., and Schauser, K. 1999. Javelin: Parallel computing on the internet. *Future Generation Computer Systems, Special Issue on Metacomputing 15,* 5-6 (Oct.), 659–674.

Nisan, N., London, S., Regev, O., and Camiel, N. 1998. Globally distributed computation over the internet-the popcorn project. In *Proceedings of the 18th International Conference on Distributed Computing Systems*. IEEE CS Press, Amsterdam, Netherlands, 592–601.

Obreiter, P. and Nimis, J. 2003. A taxonomy of incentive patterns: The design space of incentives for cooperation. In *Proceedings of the 2th International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2003)*. LNCS 2872, Springer-Verlag, Melbourne, Australia, 89–100.

Ping, T. T., Sodhy, G. C., Yong, C. H., Haron, F., and Buyya, R. 2001. A market-based scheduler for jxta-based peer-to-peer computing system. In *Proceedings of International Conference on Computational Science and its Applications (ICCSA 2004)*. LNCS 3046, Springer-Verlag, Perugia, Italy, 147–157.

Ranganathan, K., Ripeanu, M., Sarin, A., and Foster, I. 2004. Incentive mechanisms for large collaborative resource sharing. In *Proceedings of the 4th IEEE International symposium on Cluster Computing and the Grid (CCGRID 2004)*. IEEE, Chicago, USA, 1–8.

Renaud, C. G. and Playez, N. 2003. Result checking in global computing systems. In *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS'03)*. ACM, San Francisco, USA, 226–233.

Roehrig, M., Ziegler, W., and Wieder, P. 2002. Grid scheduling dictionary of terms and keywords. *GFD-I.11, Grid Scheduling Dictionary WG (SD-WG)*.

Rotithor, H. G. 1994. Taxonomy of dynamic task scheduling schemes in distributed computing system. *IEE Proceedings Computers and Digital Techniques 141,* 1 (Jan.), 1–10.

Sarmenta, L. F. G. 2001. Volunteer computing. Ph.D. thesis, MIT, Cambridge, USA.

Sarmenta, L. F. G. 2002. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems 18,* 4 (Mar.), 561–572.

Sarmenta, L. F. G. and Hirano, S. 1999. Bayanihan: building and studying web-based volunteer computing systems using java. *Future Generation Computer Systems, Special Issue on Metacomputing 15,* 5-6 (Oct.), 675–686.

SETI@home. *http://setiathome.ssl.berkeley.edu.*

Shivaratri, N. G., Krueger, P., and Singhal, M. 1992. Load distributing for locally distributed systems. *IEEE Computer 25,* 12 (Dec.), 33–44.

Shopf, J. M. 2003. Ten actions when grid scheduling. In *Grid Resource Management: Sate of the Art and Future Trends*. J. NABRZYSKI, J. M. SCHOPF and J. WEGLARZ Eds. Kluwer Academic, 15–24.

Sonnek, J., Chandra, A., and Weissman, J. B. 2007. Adaptive reputation-based scheduling on unreliable distributed infrastructures. *IEEE Transactions on Parallel and Distributed Systems 18,* 11 (Nov.), 1551–1564.

Steinmetz, R. and Wehrle, K. 2005. *Peer-to-Peer Systems and Applications*. Springer-Verlag, Berlin, Germany.

Subramanian, R. and Goodman, B. D. 2005. *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*. IDEA Group Publishing, Hershey, USA.

Tannenbaum, T., Wright, D., Miller, K., and Livny, M. 2003. Condor-a distributed job scheduler. In *Beowulf Cluster Computing with Linux*. W. Gropp, E. Lusk and T. Sterling Eds. The MIT Press, 379–426.

Taufer, M., Anderson, D., Cicotti, P., and III, C. L. B. 2005. Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Heterogeneous Computing Workshop (HCW'05)*. IEEE CS Press, Denver, USA, 119a.

Thain, D., Tannenbaum, T., and Livny, M. 2003. Condor and the grid. In *Grid Computing : Making the Global Infrastructure a Reality*. F. Berman, G. Fox, and A. J.G. Hey Eds. Wiley, Chichester, West Susses, 299–336.

Thain, D., Tannenbaum, T., and Livny, M. 2005. Distributed computing in practice : The condor experience. *Concurrency and Computation: Practice and Experience 17,* 2-4 (Feb.), 323–356.

Tsaregorodtsev, A., Garonne, V., and Stokes-Rees, I. 2004. Dirac: a scalable lightweight architecture for high throughput computing. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID2004)*. IEEE CS Press, Pittsburgh, USA, 15–24.

Venugopal, S., Buyya, R., and Ramamohanarao, K. 2006. A taxonomy of data grids for distributed data sharing, management and processing. *ACM Computing Surveys 38,* 1 (Mar.), 1–53.

Yeo, C. S. and Buyya, R. 2006. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience 36,* 13 (Nov.), 1381–1419.

Yu, J. and Buyya, R. 2005. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record, Special Issue on Scientific Workflows 34,* 3, 44–49.

Zhao, S. and Lo, V. 2005. Result verification and trust-based scheduling in open peer-to-peer cycle sharing systems. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*. IEEE CS Press, Konstanz, Germany, 31–38.

Zhong, L., Wen, D., Ming, Z. W., and Peng, Z. 2003. Paradropper: a general-purpose global computing environment built on peer-to-peer overlay network. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003), Workshop on New Advances of Web Server and Proxy Technologies (NAWSPT)*. IEEE CS Press, Providence, USA, 954–957.

Zhou, D. and Lo, V. 2004. Cluster computing on the fly: resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*. IEEE CS Press, Chicago, USA, 66–73.

Zhou, D. and Lo, V. 2005. Wave scheduler: Scheduling for faster turnaround time in peer-to-peer desktop grid systems. In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*. LNCS 3834, Springer-Verlag, Cambridge, USA, 194–218.

Zhou, S. 1988. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering 14,* 9, 1327–1341.

Zhu, Y., Xiao, L., Xu, Z., and Ni, L. M. 2006. Incentive-based scheduling in grid computing. *Concurrency and Computation: Practice and Experience 18,* 14 (Dec.), 1729–1746.