MGST: A Framework for Performance Evaluation of Desktop Grids

Majd Kokaly, Issam Al-Azzoni, Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada
kokalym@mcmaster.ca, alazzoi@mcmaster.ca, downd@mcmaster.ca

Abstract

Desktop Grids are rapidly gaining popularity as a cost-effective computing platform for the execution of applications with extensive computing needs. As opposed to grids and clusters, these systems are characterized by having a non-dedicated infrastructure. These unique characteristics need to be considered in developing resource management strategies for Desktop Grids. Several frameworks for the performance evaluation of resource management strategies have been suggested for grids. However, similar projects for Desktop Grids are still lacking. This paper presents MGST, the first performance testing framework for Desktop Grids. We discuss the design of the tool and show how it can be used to analyze and improve the performance of an existing Desktop Grid scheduling policy.

1. Introduction

Desktop Grids have emerged as an important methodology to harness the idle cycles of a large number of desktop PCs connected over the Internet or an enterprise's local area network. Such systems allow the development of applications to solve large problems and sustain throughputs far exceeding those of much more expensive supercomputers. These systems are characterized by the non-dedication of their machines. Grids and clusters, on the other hand, have a dedicated infrastructure whose size is much smaller than what has been achieved with Desktop Grids. Desktop Grids have recently received a lot of attention because of the success of several popular applications such as SETI@home [23].

A Desktop Grid employs a scheduling policy responsible for assigning tasks to resources in order to optimize certain performance requirements. Several scheduling policies have been suggested for Desktop Grids (see Choi *et al.* [10]). A scheduling policy for Desktop Grids must support systems with a very large number of machines. Also, the policy needs to cope with the high volatility and non-dedication of resources. Further adding to the complexity of scheduling for Desktop Grids is the inherent heterogeneity of such systems. Addressing these issues presents unique challenges to the design of effective scheduling policies for Desktop Grids.

This paper presents the McMaster Grid Scheduling Testing (MGST) framework for the performance evaluation of Desktop Grid scheduling policies. To the best of our knowledge, MGST is the first performance testing framework developed specifically for Desktop Grids. Other frameworks, such as GrenchMark [13] and DiPerF [22], are designed specifically for grids and do not incorporate the unique characteristics of Desktop Grids.

A testing framework for such systems should aim to simplify and automate performance testing. The testing framework should provide mechanisms for reproducing experiments, generating required workloads, replaying realistic traces and analyzing results. Furthermore, the framework should be easily deployed on top of a distributed testbed allowing for realistic testing. Extensibility, automation, ease-of-use, flexibility and accuracy are a set of non-functional requirements on the framework design.

Performance testing over a distributed testbed is complicated due to several factors [22]. Issues such as clock synchronization, heterogeneity of resources, scalability, and resource coordination make it a difficult task to do automated performance testing. MGST, as an alternative, simplifies and automates the process and saves the tester the burden of worrying about such details.

Deploying MGST on a distributed testbed enables realistic performance testing, far more effective than the use of simulation. Given the potential size and diversity of today's Desktop Grids, simulation suffers from scalability issues. Furthermore, simulation is prone to error, and may not capture the complex dynamic behaviour present in such systems. As a case study, we use MGST to study and analyze the performance of the LPAS_DG policy (Al-Azzoni and Down [2]), a scheduling policy for Desktop Grids whose performance has to date only been analyzed using simulation. The LPAS_DG policy is described in Section 4.1. We use the results from the MGST deployment to make several recommendations for the practical application of the LPAS_DG policy.

The paper is organized as follows. Section 2 defines the workload model used by the current implementation of MGST. In Section 3, we give an overview of the MGST design. In Section 4.2, we use MGST to analyze the performance of the LPAS_DG policy. The literature related to this work is discussed in Section 5. Section 6 concludes the paper and outlines future research work.

2. Desktop Grid Model

In our model for a Desktop Grid, there is a dedicated scheduler for assigning incoming tasks to the requesting machines. The tasks are assumed to be independent and atomic. In the literature, parallel applications whose tasks are independent are sometimes referred to as Bag-of-Tasks applications (BoT) (as in Anglano *et al.* [4]) or parameter-sweep applications (as in Casanova *et al.* [8]). Such applications have been observed to be the predominant applications in large-scale distributed computing systems, such as Desktop Grids (see Iosup *et al.* [15]).

The scheduler applies a pull-based scheduling policy (see Choi *et al.* [9, 10]). In pull-based scheduling, when a machine becomes available, it sends a request to the scheduler in order to be assigned a new task for execution. Using pull-based scheduling is necessary due to the property that the machines are not dedicated in Desktop Grids. One of the results of using pull-based scheduling is that tasks queue at the scheduler side. There is no queueing at the machines; in fact, in Desktop Grids, one machine executes at most one task at a time without preemption (see [10], Domingues *et al.* [11], and Kondo *et al.* [18]). Also, in pull-based scheduling, the scheduler makes a decision as soon as it receives a request from a machine [10].

In Desktop Grids, machines can fail (or become unavailable) at any time without any advance notice [4]. If a machine fails while executing a task, then that task needs to be resubmitted to the scheduler. We assume that the Desktop Grid is mainly used to execute short-lived applications [18]. Hence, in such systems, we do not consider fault tolerant scheduling mechanisms such as checkpointing, migration and replication, due to their overhead.

One of the basic properties of Desktop Grids is the non-dedication of machines. When a machine is available, it may also run local jobs (*i.e.*, jobs submitted by a local user). The machines' local jobs are always given higher priority. When a machine is busy with local jobs, the result is a slowing down of the execution of the Desktop Grid tasks submitted by the scheduler to the machine. As in [4, 18], we assume that once a task is submitted to a machine, the task can not be resubmitted unless a failure occurs.

To describe the availability and dedication of a machine for the execution of the Desktop Grid tasks, we distinguish between machine and CPU availability (see Kondo *et al.* [19] and Nurmi *et al.* [21]). The former is a binary value that indicates if the machine is reachable. Examples of machine unavailability include power failure or machine reboot. The latter is a percentage value that quantifies the fraction of the CPU that can be exploited by Desktop Grid applications.

3. The MGST Tool

In this section we discuss MGST, first stating the objectives and usage of the tool and then illustrating the design. The tool was introduced in [17].

3.1. Objectives and Usage

Scheduling schemes are based on a specific workload model and a set of assumptions. Performance is typically predicted using mathematical analysis and/or simulation-based experiments. The actual performance, however, can be quite different. There are two reasons for that; the first one is the fact that the assumptions made during the design phase are usually incorporated in the simulation tool developed by the researchers to test the schemes. The second reason is that some factors that are not taken into consideration in the theoretical models can have considerable impact on the performance of the scheme.

MGST was developed to serve as a test bed for scheduling schemes. As opposed to simulation tools, MGST provides testers with realistic conditions.

Having such a testing environment allows researchers to:

- Verify that a scheduling policy can be implemented.
- Verify that the scheduling policy is behaving as it was intended to.
- Verify that the assumptions made actually hold and are reasonable.
- Determine potential areas for improvement of a particular scheduling policy.

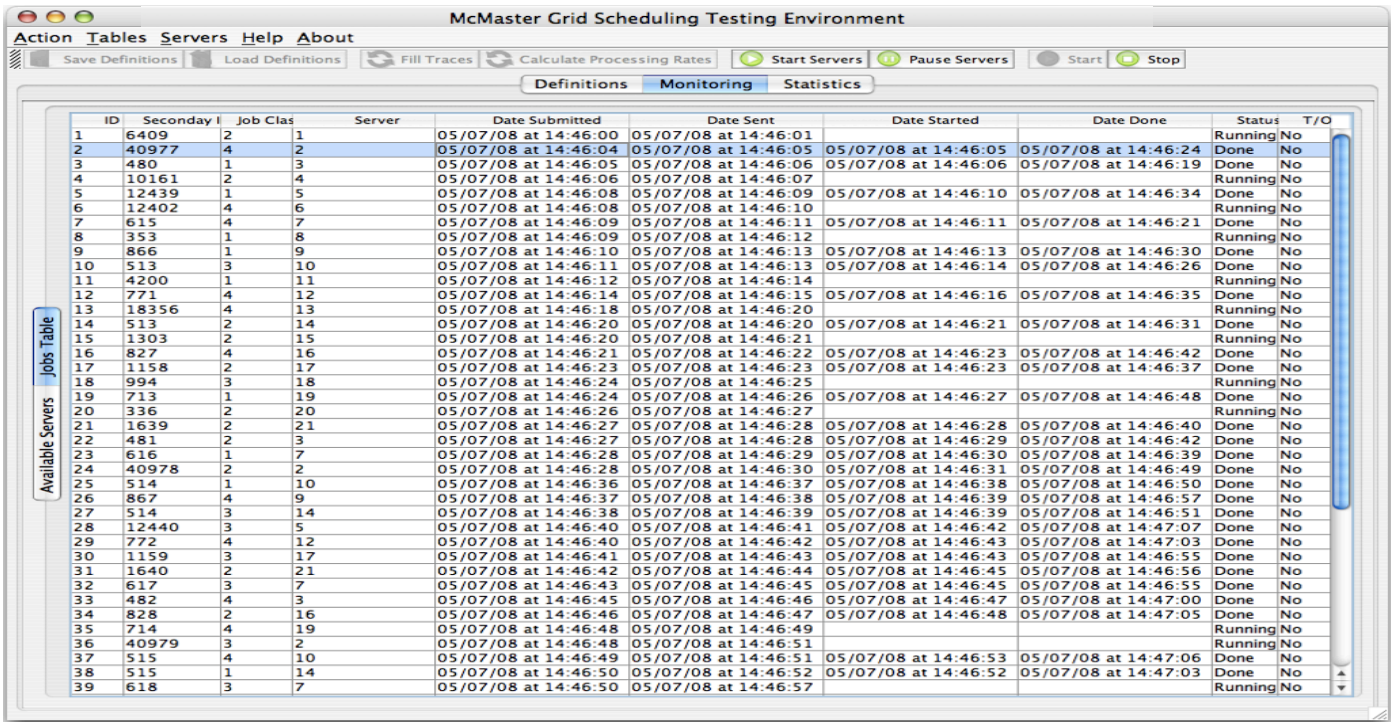
To conduct an experiment, a collection of networked computers is used. One of these computers is chosen to be the central machine (the mapper) that will run the MGST tool. The remaining machines (servers) run only the server module (puller) of the MGST software.

An experiment is initiated on the mapper using the MGST interface, and the mapper starts mapping jobs to the servers. Servers execute the jobs, returning the results back when completed. During an experiment, the MGST software at the mapper records the events. The tester is also able to monitor the experiment using the monitoring capabilities of the MGST. For example, during an experiment testers can view the current jobs queued on the mapper, the job status and various system-level parameters such as average waiting time for a specific class of jobs. When the experiment is completed, MGST can produce files readable by spread sheet programs. Figure 3.1 shows a screen shot of the MGST.

Log files, that are produced, record all of the events of a conducted experiment. The tool also generates various parameters characterizing the performance of a policy. The Log files are produced by the software running on the mapper. The results are stored in text files that can be read by spread sheet programs. MGST is currently limited to Macintosh-based computers. Extending the support for more architectures is one of our future plans.

3.2. Design

Extensibility was the main software quality desired in the tool and therefore was the main design goal. The notion of extensibility in the case of MGST is that the tool and its features can be extended by adding scheduling schemes to be tested in an



ID	Seconday I	Job Clas	Server	Date Submitted	Date Sent	Date Started	Date Done	Status	T/O
1	6409	2	1	05/07/08 at 14:46:00	05/07/08 at 14:46:01			Running	No
2	40977	4	2	05/07/08 at 14:46:04	05/07/08 at 14:46:05	05/07/08 at 14:46:05	05/07/08 at 14:46:24	Done	No
3	480	1	3	05/07/08 at 14:46:05	05/07/08 at 14:46:06	05/07/08 at 14:46:06	05/07/08 at 14:46:19	Done	No
4	10161	2	4	05/07/08 at 14:46:06	05/07/08 at 14:46:07			Running	No
5	12439	1	5	05/07/08 at 14:46:08	05/07/08 at 14:46:09	05/07/08 at 14:46:10	05/07/08 at 14:46:34	Done	No
6	12402	4	6	05/07/08 at 14:46:08	05/07/08 at 14:46:10			Running	No
7	615	4	7	05/07/08 at 14:46:09	05/07/08 at 14:46:11	05/07/08 at 14:46:11	05/07/08 at 14:46:21	Done	No
8	353	1	8	05/07/08 at 14:46:09	05/07/08 at 14:46:12			Running	No
9	866	1	9	05/07/08 at 14:46:10	05/07/08 at 14:46:13	05/07/08 at 14:46:13	05/07/08 at 14:46:30	Done	No
10	513	3	10	05/07/08 at 14:46:11	05/07/08 at 14:46:13	05/07/08 at 14:46:14	05/07/08 at 14:46:26	Done	No
11	4200	1	11	05/07/08 at 14:46:12	05/07/08 at 14:46:14			Running	No
12	771	4	12	05/07/08 at 14:46:14	05/07/08 at 14:46:15	05/07/08 at 14:46:16	05/07/08 at 14:46:35	Done	No
13	18356	4	13	05/07/08 at 14:46:18	05/07/08 at 14:46:20			Running	No
14	513	2	14	05/07/08 at 14:46:20	05/07/08 at 14:46:20	05/07/08 at 14:46:21	05/07/08 at 14:46:31	Done	No
15	1303	2	15	05/07/08 at 14:46:20	05/07/08 at 14:46:21			Running	No
16	827	4	16	05/07/08 at 14:46:21	05/07/08 at 14:46:22	05/07/08 at 14:46:23	05/07/08 at 14:46:42	Done	No
17	1158	2	17	05/07/08 at 14:46:23	05/07/08 at 14:46:23	05/07/08 at 14:46:23	05/07/08 at 14:46:37	Done	No
18	994	3	18	05/07/08 at 14:46:24	05/07/08 at 14:46:25			Running	No
19	713	1	19	05/07/08 at 14:46:24	05/07/08 at 14:46:26	05/07/08 at 14:46:27	05/07/08 at 14:46:48	Done	No
20	336	2	20	05/07/08 at 14:46:26	05/07/08 at 14:46:27			Running	No
21	1639	2	21	05/07/08 at 14:46:27	05/07/08 at 14:46:28	05/07/08 at 14:46:28	05/07/08 at 14:46:40	Done	No
22	481	2	3	05/07/08 at 14:46:27	05/07/08 at 14:46:28	05/07/08 at 14:46:29	05/07/08 at 14:46:42	Done	No
23	616	1	7	05/07/08 at 14:46:28	05/07/08 at 14:46:29	05/07/08 at 14:46:30	05/07/08 at 14:46:39	Done	No
24	40978	2	2	05/07/08 at 14:46:28	05/07/08 at 14:46:30	05/07/08 at 14:46:31	05/07/08 at 14:46:49	Done	No
25	514	1	10	05/07/08 at 14:46:36	05/07/08 at 14:46:37	05/07/08 at 14:46:38	05/07/08 at 14:46:50	Done	No
26	867	4	9	05/07/08 at 14:46:37	05/07/08 at 14:46:38	05/07/08 at 14:46:39	05/07/08 at 14:46:57	Done	No
27	514	3	14	05/07/08 at 14:46:38	05/07/08 at 14:46:39	05/07/08 at 14:46:39	05/07/08 at 14:46:51	Done	No
28	12440	3	5	05/07/08 at 14:46:40	05/07/08 at 14:46:41	05/07/08 at 14:46:42	05/07/08 at 14:47:07	Done	No
29	772	4	12	05/07/08 at 14:46:40	05/07/08 at 14:46:42	05/07/08 at 14:46:43	05/07/08 at 14:47:03	Done	No
30	1159	3	17	05/07/08 at 14:46:41	05/07/08 at 14:46:43	05/07/08 at 14:46:43	05/07/08 at 14:46:55	Done	No
31	1640	2	21	05/07/08 at 14:46:42	05/07/08 at 14:46:44	05/07/08 at 14:46:45	05/07/08 at 14:46:56	Done	No
32	617	3	7	05/07/08 at 14:46:43	05/07/08 at 14:46:45	05/07/08 at 14:46:45	05/07/08 at 14:46:55	Done	No
33	482	4	3	05/07/08 at 14:46:45	05/07/08 at 14:46:46	05/07/08 at 14:46:47	05/07/08 at 14:47:00	Done	No
34	828	2	16	05/07/08 at 14:46:46	05/07/08 at 14:46:47	05/07/08 at 14:46:48	05/07/08 at 14:47:05	Done	No
35	714	4	19	05/07/08 at 14:46:48	05/07/08 at 14:46:49			Running	No
36	40979	3	2	05/07/08 at 14:46:48	05/07/08 at 14:46:51			Running	No
37	515	4	10	05/07/08 at 14:46:49	05/07/08 at 14:46:51	05/07/08 at 14:46:53	05/07/08 at 14:47:06	Done	No
38	515	1	14	05/07/08 at 14:46:50	05/07/08 at 14:46:52	05/07/08 at 14:46:52	05/07/08 at 14:47:03	Done	No
39	618	3	7	05/07/08 at 14:46:50	05/07/08 at 14:46:57			Running	No

Figure 1. General Screen Shot

easy manner or by adding features or introducing modifications based on researchers' needs.

Certain software qualities such as simplicity and modularity were maintained leading to the design of an extensible tool. The tool was divided into seven modules each having a defined responsibility.

Adjuster: The purpose of this module is to impose artificial properties on the system. It is used to impose heterogeneity.

Mapper: As its name indicates, this module does the actual mapping of the jobs.

Job Generator: This module is responsible for generating jobs and sending them to the Mapper module.

Logger: This module keeps a record of the events that happen during the course of a test.

Executer: This module is responsible for executing and managing the tasks on assigned machines.

Puller: Unlike all of the other modules, this module is deployed at the servers. The Puller is responsible for maintaining availability information and sending it to the central Mapper notifying it when servers become available. In addition, the Puller notifies the Mapper when a job is completed.

User Interface: This module is responsible for providing the graphical user interface to users.

In addition, four forms of documentation were introduced to aid future developers in modifying and extending the tool. These forms of documentation are Javadoc for the implemented classes, comments throughout the code, a design document and a user manual.

4. Experimental Results

In this section we use MGST to analyze the performance of the LPAS_DG policy, a dynamic scheduling policy for Desktop Grids [2]. First, we give a brief overview of the policy.

4.1. The LPAS_DG Policy

The Linear Programming Based Affinity Scheduling policy for Desktop Grids (LPAS_DG) is designed for heterogeneous Desktop Grids that execute multiple applications. In such systems, one can think of an application (or, a class) as consisting of tasks whose expected execution times on a given machine are the same. Let $\mu'_{i,j}$ be the nominal execution rate for tasks of class i at machine j, hence $1/\mu'_{i,j}$ is the mean nominal execution time for class i tasks at machine j. Assume that there are N classes of tasks. Tasks that belong to the same class i have arrival rate α_i. Let α be the arrival rate vector, the ith element of α is α_i. Let the number of machines in the system be M.

In addition to the arrival and execution rates, the policy exploits information on CPU availabilities. The policy assumes that when a machine becomes available, it sends a request for a new task to the scheduler. As in [4], we assume that the machine also supplies the expected proportion of time that it is going to spend in executing the Desktop Grid tasks during its coming availability period (*i.e.,* its CPU availability). Thus, we can define the effective execution rate $\mu_{i,j}$ for the submitted tasks as:

$$\mu_{i,j} = \mu'_{i,j} \times a_j$$

where a_j represents the fraction of machine j's capacity that is available for executing the Desktop Grid tasks during its coming

availability period. Also, let μ be the effective execution rate matrix, having (i, j) entry $\mu_{i,j}$.

The policy requires solving the following allocation LP (Andradóttir *et al.* [3]) at each machine availability/unavailability event, where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$. The variables $\delta_{i,j}$ are to be interpreted as the proportional allocation of machine j to class i.

$$\max \ \lambda$$

$$\text{s.t.} \sum_{j=1}^{M} \delta_{i,j}\mu'_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N, \tag{1}$$

$$\sum_{i=1}^{N} \delta_{i,j} \leq a_j, \qquad \text{for all } j = 1, \ldots, M, \tag{2}$$

$$\delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \text{ and } j = 1, \ldots, M. \tag{3}$$

The left-hand side of (1) represents the total execution capacity assigned to class i by all machines in the system. The right-hand side represents the arrival rate of tasks that belong to class i scaled by a factor of λ. Thus, (1) enforces that the total capacity allocated for a class should be at least as large as the scaled arrival rate for that class. The constraint (2) prevents overallocating a machine and (3) states that negative allocations are not allowed.

Let λ^* and $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, be an optimal solution to the allocation LP. The allocation LP always has a solution, since no lower bound constraint is put on λ. Let δ^* be the machine allocation matrix where the (i, j) entry is $\delta^*_{i,j}$.

Whenever a machine becomes available or unavailable, the scheduler solves the allocation LP to find $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. If a machine j becomes unavailable, then $a_j = 0$. In this case, $\delta^*_{i,j} = 0$ for $i = 1, \ldots, N$. On the other hand, if a machine j becomes available, a_j is equal to the predicted CPU availability for machine j during its next expected machine availability period.

The LPAS_DG policy is defined as follows. When a machine j requests a task, let S_j denote the set of task classes i such that $\delta^*_{i,j}$ is not zero ($S_j = \{i : \delta^*_{i,j} \neq 0\}$). Let $D_i(t)$ be the waiting time (sojourn time) of the head of the line class i task at the time t of making the scheduling decision. The scheduler assigns machine j the longest-waiting (head of the line) class i task such that

$$\mu_{i,j}\delta^*_{i,j} > 0 \text{ and } i \in \arg\max_i \mu_{i,j} D_i(t).$$

Note that the LPAS_DG policy does not use the actual values for $\{\delta^*_{i,j}\}$, beyond differentiating between the zero and nonzero elements. Regardless, we must solve the allocation LP to know where the zeros are.

4.2. Experimental Results

We used MGST to analyze the performance of the LPAS_DG policy under realistic conditions. We tested the scheme on several systems. Each test was conducted two times, once using the simulation tool used in [2] and once with MGST. The metric used in the simulations and experiments is the average response time, including average communication delay for the MGST experiments. The communication delay is the difference between the time a job is sent to be executed and the time it begins execution. This delay occurs mainly due to network communication delays, but it could also be caused by the software layer responsible for the distribution and execution of the tasks.

The experiments were conducted on four categories of systems depending on machine and job heterogeneity. Machine heterogeneity refers to the average variation in the *rows* of the execution matrix μ. On the other hand, job heterogeneity refers to the average variation of the *columns*. Based on Armstrong [5], we define the following categories for heterogeneity:

- High job heterogeneity and high machine heterogeneity (*HiHi*).
- High job heterogeneity and low machine heterogeneity (*HiLo*).
- Low job heterogeneity and high machine heterogeneity (*LoHi*).
- Low job heterogeneity and low machine heterogeneity (*LoLo*).

Two to four experiments were conducted on each category. In some experiments failures were enabled meaning that machines can fail while executing jobs. Machines were in some experiments fully dedicated ($a_j = 1$ for all j), where their full resources were used exclusively by the desktop grid. In other experiments only a percentage of the resources were available for the grid. We will use the following acronyms to express these properties in the experiments: FE, FD, MFD, MPD for failures enabled, failures disabled, machine fully dedicated and machines partially dedicated respectively. We will also use Mn to denote the n^{th} machine (e.g. M1 is machine 1). Similarly, we define Gn for groups of machines.

4.2.1 HiHi

The experiments in this category were conducted on 6 machines and 4 classes of jobs. Table 1 shows the execution rates. The arrival rates of the job classes were: $\alpha = \begin{bmatrix} 2.25 & 4.50 & 7.20 & 12.60 \end{bmatrix}$.

Class	M1	M2	M3	M4	M5	M6
1	2.0	2.0	2.0	2.0	2.0	2.0
2	1.0	20.0	3.7	7.1	2.4	8.7
3	1.0	20.0	9.4	3.7	7.2	2.7
4	1.0	20.0	2.8	5.9	4.4	6.3

Table 1. Execution Rates of Setting HiHi

The average response time for each class of jobs and the over all average response time are shown in Table 2. The simulation results in this and all the following tables are at a 95% confidence interval.

	MFD/FD		MPD/FD		MFD/FE		MPD/FE	
Class	Sim	MGST	Sim	MGST	Sim	MGST	Sim	MGST
1	(0.56, 0.57)	0.58	(0.97, 0.98)	0.99	(0.61, 0.61)	0.61	(1.10, 1.11))	1.03
2	(0.33, 0.34)	0.34	(0.22, 0.22)	0.37	(0.35, 0.35)	0.45	(1.10, 1.11)	0.50
3	(0.18, 0.18)	0.22	(0.27, 0.27)	0.29	(0.19, 0.20)	0.20	(0.32, 0.32)	0.46
4	(0.11, 0.11)	0.17	(0.16, 0.16)	0.43	(0.13, 0.13)	0.15	(0.26, 0.27)	1.36
Overall	(0.20, 0.21)	0.24	(0.27, 0.27)	0.42	(0.23, 0.23)	0.25	(0.35, 0.36)	0.94

Table 2. Results of Experiment on HiHi Setting

In the MPD/FD and MPD/FD experiments M4, M5 and M6 had availability $a_j = 0.5$. The remaining machines were fully dedicated ($a_j = 1$). In the MPD/FE and MPD/FE experiments each machine failed at the rate 0.05 per time-unit and the mean fault time was 2 time-units. The periods were exponentially distributed.

In experiment MPD/FD the actual performance of the LPAS_DG policy was worse than the simulation had predicated. The reason is discussed in Section 4.3.

4.2.2 LoHi

This setting was constructed from 21 machines and 4 job classes. This setting was from category *HiLo*. There were seven groups of machines. Members of the same group have the same execution rates. Machines in group 1 are machines 1, 8 and 15, machines in group 2 are machines 2, 9 and 16, etc. The arrival rates of the job classes were: $\alpha = \begin{bmatrix} 22.5 & 22.5 & 18.0 & 18.0 \end{bmatrix}$.

Class	G1	G2	G3	G4	G5	G6	G7
1	2.20	7.00	10.25	1.00	5.70	0.50	12.00
2	1.95	7.05	9.78	0.95	5.65	0.56	11.85
3	2.00	7.25	10.02	0.98	5.75	0.67	11.80
4	2.05	6.75	9.99	1.02	5.82	0.49	12.05

Table 3. Execution Rates of Setting LoHi

The average response time for each class of jobs and the over all average response time are shown in Table 4.

	MFD/FD		MPD/FD	
Class	Sim	MGST	Sim	MGST
1	(0.22, 0.22)	0.31	(0.24, 0.24)	0.36
2	(0.12, 0.12)	0.22	(0.13, 0.13)	0.26
3	(0.30, 0.30)	0.37	(0.37, 0.37)	0.44
4	(0.29, 0.29)	0.35	(0.35, 0.35)	0.47
Overall	(0.22, 0.22)	0.31	(0.26, 0.27)	0.37

Table 4. Results of Experiment on LoHi Setting

In the MPD/FD experiment M2, M11 and M19 the availability $a_j = 0.5$. M3, M12, M20 had availability $a_j = 0.75$. The remaining machines were fully dedicated ($a_j = 1$).

The average response times obtained by MGST were within a reasonable range from the results obtained by simulation. The average response times of the MGST experiment were slightly higher due to the fact that actual processing rates were slower. This is discussed in Section 4.3.

4.2.3 HiLo

This setting was constructed from 21 machines and 4 job classes divided into seven groups in the same way machines in the setting LoHi were divided. Execution rates are shown in Table 5. G1 to G7 are group 1 to group 7. The arrival rates of the job classes were: $\alpha = \begin{bmatrix} 10.50 & 21.00 & 26.25 & 26.25 \end{bmatrix}$.

Class	G1	G2	G3	G4	G5	G6	G7
1	2.00	2.50	2.25	2.00	2.20	1.75	2.25
2	4.50	4.0	4.20	4.00	3.80	3.90	3.95
3	6.00	6.20	6.25	6.00	5.75	5.90	6.05
4	10.00	10.25	10.50	9.50	10.25	10.25	10.00

Table 5. Execution Rates of Setting HiLo

The average response time for each class of jobs and the over all average response time are shown in Table 6. The availabilities of machines were as in Table 6.

	MFD/FD		MPD/FD	
Class	Sim	MGST	Sim	MGST
1	(0.49, 0.49)	0.50	(0.79, 0.80)	1.22
2	(0.28, 0.28)	0.31	(0.42, 0.42)	0.77
3	(0.24, 0.24)	0.32	(0.27, 0.27)	0.53
4	(0.14, 0.14)	0.35	(0.19, 0.19)	0.73
Overall	(0.25, 0.25)	0.35	(0.35, 0.35)	0.74

Table 6. Results of Experiment on HiLo Setting

In the MPD/FD experiment M2, M11 and M19 had availability $a_j = 0.5$. M3, M12, M20 had availability $a_j = 0.75$. The remaining machines were fully dedicated ($a_j = 1$).

Compared to simulation, LPAS_DG performed poorly in the MGST experiment. The reason is that the ideal overall load on the machines was fairly high (86.4%), but the different sources of errors and overhead caused the actual load to be close to 100%. The sources of errors are higher overall arrival rates, over estimation for processing rates and communication overhead coupled with the scheduling delay. See Section 4.3 for more details.

4.2.4 LoLo

This setting was constructed from 21 machines and 4 job classes divided into seven groups in the same way machines in the setting LoHi were divided. Execution rates are shown in Table 7. G1 to G7 are group 1 to group 7. The arrival rates of the job classes were: $\alpha = \begin{bmatrix} 18.00 & 20.25 & 15.75 & 22.50 \end{bmatrix}$.

Class	G1	G2	G3	G4	G5	G6	G7
1	5.00	5.05	4.95	4.98	4.70	5.20	5.25
2	5.25	5.09	4.90	4.92	5.00	5.13	5.14
3	4.45	5.00	4.90	4.45	4.90	5.00	5.10
4	5.02	4.95	5.00	5.02	5.25	4.75	5.00

Table 7. Execution Rates of Setting LoLo

This experiment included machine failures. The mean uptime was 50 time units and the mean failure period was 2 time units. The periods were exponentially distributed.

The mean response time for each class of jobs and the over all response time are shown in Table 8.

	MFD/FD		MPD/FD		MFD/FE		MPD/FE	
Class	Sim	MGST	Sim	MGST	Sim	MGST	Sim	MGST
1	(0.25, 0.25)	0.27	(0.28, 0.28)	0.39	(0.25, 0.25)	0.35	(0.31, 0.31)	0.52
2	(0.23, 0.23)	0.28	(0.30, 0.30)	0.39	(0.24, 0.24)	0.34	(0.32, 0.32)	0.63
3	(0.23, 0.23)	0.28	(0.27, 0.27)	0.35	(0.24, 0.24)	0.33	(0.32, 0.32)	0.57
4	(0.21, 0.22)	0.25	(0.32, 0.32)	0.36	(0.24, 0.24)	0.29	(0.34, 0.34)	0.52
Overall	(0.23, 0.23)	0.27	(0.30, 0.30)	0.37	(0.24, 0.24)	0.33	(0.32, 0.32)	0.56

Table 8. Results of Experiment on LoLo Setting

In the MPD/FD and MPD/FD experiments M2, M11 and M19 had availability $a_j = 0.5$. M3, M12 and M20 had availability $a_j = 0.5$. The remaining machines were fully dedicated ($a_j = 1$). In the MPD/FE and MPD/FE each machine failed at the rate 0.05 per time-unit and the mean fault time was 2 time-units. The periods were exponentially distributed.

The response times in the results of our experiment were significantly higher than the simulation results. The reason behind this is the high load coupled with failures and over estimation of the execution rates (the assumed execution rates were higher than the actual ones in this experiment). See Section 4.3.

4.3. Analysis

4.3.1 Modifications

The LPAS_DG scheduling policy was implemented for the first time in our testing environment. Here we give a few remarks regarding the implementation of this policy. The MGST tool allowed us to study implementability (To Do), a value (To Do) and of itself.

The LPAS_DG policy is silent on how to choose a server if there is more than one available to serve a job. For simplicity, in our first implementation we chose the FCFS policy to choose servers, but this resulted in performance degradation. The performance is affected because the scheduling process might be blocked when the head of the available servers queue is a server capable of executing a limited number of job classes and none of the currently queued jobs belong to any of these classes. The FCFS implementation was modified to remove the head of the server queue and insert it at the back of the queue, if there are jobs in the jobs queue but this server is not able to execute any of them. However, we believe that the performance of the LPAS_DG can be further improved by employing a suitable policy to choose servers from the available servers queue, especially in the case of a low or medium load on the system. We believe that further research must be conducted to come up with a suitable policy. However, we recommend the LPAS scheduling policy for clusters (Al-Azzoni and Down [1]) to be considered as a possible solution, since this policy is suitable for choosing servers for jobs in heterogeneous environments. This modification is not necessary but could improve the performance under medium to low loads.

LPAS_DG makes decisions based the matrix δ^*, which is produced by solving a linear programming problem (Section 4.1). The δ^* matrix depends on the values of a_j. As a result, in [2] it is suggested that a new δ^* matrix must be produced at every availability/unavailability event (Section 4.1).

Since the matrix δ^* depends on a_j, and the machines' a_j varies between availability and unavailability events, we think that δ^* should be updated every time any a_j changes. This solution is expensive to implement because it is very hard to notify the mapper of every change to every a_j. In addition, this will require solving the allocation LP frequently, which is also expensive and will raise a scalability problem. To solve this issue, we assumed a time resolution T_{system} (The mapper could become the bottleneck). Here, the values of a_j are sent to the mapper periodically, and it solves the allocation LP after receiving the updated values of a_j. The determination of an optimal update period is open to research. We believe that this modification is necessary to make LPAS_DG scalable.

4.3.2 Robust Modifications

In some experiments the performance of the scheduling schemes differed from the simulation results due to the machines experiencing unexpectedly high loads. The different sources of error that can occur in a real system can significantly raise the load, even potentially causing instability in the system. These errors can be caused by:

1. **The actual arrival rate being larger than the assumed one.**
2. **Overestimation of processing rates.**
3. **Overhead caused by communication and scheduling delays.** Assume that a server announces its availability at time t_1, then the mapper learns of the availability of this server at time t_2 and consequently performs the scheduling and chooses a job at time t_3 and then sends the job. The server then receives the job and starts the execution at time t_4. At time t_5 the server finishes executing the job but only at time t_6 does the mapper learn that the job is done, obtaining the results at t_7. In the model, the processing time is considered to be $t_6 - t_5$, but in the actual implementation, there is an overhead of $(t_5 - t_1) + (t_7 - t_6)$. This overhead affects the load on the system if $t_6 - t_5$ is small compared to the overhead.
4. **Machine failures.** Although machines failure can be incorporated in the workload models, they can still increase the effective load due to the fact that it takes time for the mapper to realize that a server is down. This time is wasted and effectively increases the load. For example, when using LPAS_DG, suppose that server 3 is the only server executing jobs from class 1, and the execution time is 5 minutes. If server 3 fails when executing a particular job and the "time-out" parameter was set to 3 times (i.e. 3 times the estimated execution time should elapse before considering the job "timed out"), then the Mapper will not consider server 3 down until 15 minutes have elapsed from the moment that the job was sent. These 15 minutes are essentially lost, with arriving jobs from class 1 accumulating in the queue at the Mapper within that time.

If any or all of the above factors cause a significant increase in the load, the performance of the scheduling scheme will deteriorate. Note that these factors were only discovered upon

deploying LPAS_DG on MGST. They were not discovered in simulations.

The LPAS_DG policy suffered in some cases in the experiments from the above factors due to the aggressive nature of this policy in minimizing the number of machines to execute each job class. This results in exclusivity of machines for certain job classes. When one class can be executed by a small number of machines, then the performance depends only on these machines, so the effect of the factors mention above is magnified. Contrast this with FCFS, where if a machine under performs, the effect is less obvious since this under performing machine can get help from other (potentially over performing) machines. Finally, the scheduling delay can contribute to the time needed to process jobs, effectively raising the load on machines for all policies.

As a result of the MGST experiments, we propose the following suggestions to improve robustness of LPAS_DG:

1. **Arrival rates estimation improvement.** Since the LPAS_DG scheme depends on solving the allocation LP and that in turn depends on values that include arrival rates of job classes, estimates should be as accurate as possible. To do so, we propose that the actual arrival rates should be monitored (a feature that our tool provides), and check the values against the estimated values every specific time ($T_{arrival\_rate}$) and resolve the LP if one of the actual values differs from its estimate by a specific threshold percentage ($Th_{arrival\_rate}$) that depends on the load and the job class. $T_{arrival\_rate}$ could be a specific time period or a number of job arrivals from a class (e.g. 10 jobs). We believe that this solution is not computationally costly, since the checking operation requires $O(N)$ time and $O(1)$ space. We expect the number of job classes (N) to be relatively small, so there should be no scaling issues. An alternative solution is to over estimate the arrival rates of classes, however, caution must be taken to guarantee that the system is theoretically stable.

2. **Avoiding processing rates overestimation.** We propose that every processing rate entry (for a specific server for a specific job class) is modified then checked (against the estimated peer) whenever a job is done, then the LP is resolved if that entry differs from the estimated one by a specific threshold percentage ($T_{processing\_rate}$) that depends on the load and the job class. This solution requires $O(NM)$ space and $O(1)$ time. Alternatively, the processing rates can be assumed slower than they are estimated to be in a manner that guarantees that they can never be over estimated. However, caution must be taken to assure that the system is theoretically stable.

3. **Lessen the effect of communication and scheduling delays.** Let $p_{i,j}$ be an estimate of the value

$$\frac{1/\mu_{i,j}}{1/\mu_{i,j} + \tau_j} \tag{4}$$

where τ_j is the communication and scheduling delay for machine j.

In the example mentioned in point 3 in Section 4.3.2 $p_{i,j}$ would be

$$\frac{t_6 - t_5}{t_7 - t_1} \tag{5}$$

We propose that all execution rates must be multiplied by p before resolving the LP to take this effect into consideration.

4. **Lessen the machine failure effect.** We propose choosing a low value for the time out, which will result in allowing the mapper to quickly detect server failures. The downside of this approach is that the mapper might consider a server failed one when it is not.

We think that the scheduling schemes should be put to test under realistic conditions after the initial design phase. Our tool was developed for this very purpose. MGST gave us important feedback about the LPAS_DG policy. We strongly believe that our tool will aid in improving scheduling schemes in general.

5. Related Work

Our work on developing MGST relates to work on performance testing tools for large distributed systems. Recently, much attention has been given to tools for performance testing in real grids. GrenchMark [13] and DiPerF [22] are two frameworks developed specifically for the performance testing of grids. To the best of our knowledge, there is no testing framework specific to Desktop Grids. As a testing framework, MGST represents the first tool designed specifically for testing performance in Desktop Grids.

Several simulators exist to model and analyze performance in simulated environments. Simulators, such as DGSim [16] and SimGrid [7], can be used to evaluate performance of grid resource management systems. For Desktop Grids, there are several simulators, such as SimBA [24], SimBOINC [6], and DGShedSim [12]. Such tools enable simulations driven by workload and availability traces from real Desktop Grids. While the use of simulators allows the modeling of key attributes of Desktop Grids, simulators do not fully capture the dynamic behaviour in real Desktop Grids. Furthermore, issues such as scalability and diversity of Desktop Grids represent key challenges for the use of simulators.

Several of the tools mentioned above support the use of workload and availability traces. The Grid Workloads Archive [14] is a project that aims to provide workload traces collected from several production grids. The Desktop Grid Trace Archive [19] provides availability traces collected at four enterprise Desktop Grid sites. The XtremLab project [20] provides availability traces collected from large, Internet-based Desktop Grids. Similar projects are still needed to collect representative workload traces of Desktop Grid applications.

6. Conclusion and Future Work

This paper has presented MGST, a framework for the performance evaluation of resource management strategies for Desktop Grids. The framework can be easily deployed on real distributed testbeds, and is flexible and extensible. We have shown

how to use MGST to analyze and improve the performance of the LPAS_DG policy.

We are currently extending MGST with several additions:

- The ability to run real workload traces. Currently, the tool generates jobs based on configurable probability distributions. Adding the ability to generate jobs from real workload traces from actual systems will certainly add value.
- Broaden the machine types. Currently, the Executer module is implemented using an Apple technology called Xgrid. This constrains the system to use Macintosh-based computers. We plan on implementing the Executer layer in a way that will allow us to move to a richer set of platforms.

References

[1] I. Al-Azzoni and D. Down. Linear programming based affinity scheduling of independent tasks on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1671–1682, 2008.

[2] I. Al-Azzoni and D. G. Down. Dynamic scheduling for heterogeneous Desktop Grids. In *Proceedings of the 9th International Conference on Grid Computing*, pages 136–143, 2008.

[3] S. Andradóttir, H. Ayhan, and D. G. Down. Compensating for failures with flexible servers. *Operations Research*, 55(4):753–768, 2007.

[4] C. Anglano, J. Brevik, M. Canonico, D. Nurmi, and R. Wolski. Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids. In *Proceedings of the 7th International Conference on Grid Computing*, pages 56–63, 2006.

[5] R. Armstrong. Investigation of effect of different run-time distributions on SmartNet performance. Master's thesis, Naval Postgraduate School, 1997.

[6] F. Cappello, G. Fedak, D. Kondo, P. Malécot, and A. Rezmerita. Desktop grids: From volunteer distributed computing to high throughput computing production platforms. In *Handbook of Research on Scalable Computing Technologies*. IGI Global, 2009.

[7] H. Casanova, A. Legrand, and M. Quinson. SimGrid: A generic framework for large-scale distributed experiments. In *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, pages 126–131, 2008.

[8] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349–363, 2000.

[9] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang. Characterizing and classifying desktop grid. In *Proceedings of the 7th International Symposium on Cluster Computing and the Grid*, pages 743–748, 2007.

[10] S. Choi, H. Kim, E. Byun, and C. Hwang. A taxonomy of desktop grid systems focusing on scheduling. Technical Report KU-CSE-2006-1120-01, Department of Computer Science and Engeering, Korea University, November 2006.

[11] P. Domingues, A. Andrzejak, and L. Silva. Scheduling for fast turnaround time on institutional desktop grid. Technical Report TR-0027, CoreGRID, January 2006.

[12] P. Domingues, P. Marques, and L. Silva. DGSchedSim: A trace-driven simulator to evaluate scheduling algorithms for desktop grid environments. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 83–90, 2006.

[13] A. Iosup and D. Epema. GrenchMark: A framework for analyzing, testing, and comparing grids. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*, pages 313–320, 2006.

[14] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[15] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, pages 97–108, 2008.

[16] A. Iosup, O. O. Sonmez, and D. H. J. Epema. DGSim: Comparing grid resource management architectures through trace-based simulation. In E. Luque, T. Margalef, and D. Benitez, editors, *Euro-Par*, volume 5168 of *Lecture Notes in Computer Science*, pages 13–25, 2008.

[17] M. Kokaly. McMaster Grid Scheduling Testing Environment. Master's thesis, McMaster University, Canada, August 2008.

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

[19] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems*, 23(7):888–903, 2007.

[20] P. Malecot, D. Kondo, and G. Fedak. XtremLab: A system for characterizing Internet desktop grids. In *Proceedings of the 15th International Symposium on High Performance Distributed Computing*, pages 357–358, 2006.

[21] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of the 11th International Euro-Par Conference*, pages 432–441, 2005.

[22] I. Raicu, C. Dumitrescu, M. Ripeanu, and I. T. Foster. The design, performance, and use of DiPerF: An automated distributed performance evaluation framework. *Journal of Grid Computing*, 4(3):287–309, 2006.

[23] SETI@home. "http://setiathome.berkeley.edu/".

[24] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. J. Teller. SimBA: A discrete event simulator for performance prediction of volunteer computing projects. In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 189–197, 2007.