Dynamic Scheduling for Heterogeneous Desktop Grids

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

Abstract

Desktop Grids have emerged as an important methodology to harness the idle cycles of millions of participant desktop PCs over the Internet. However, to effectively utilize the resources of a Desktop Grid, it is necessary to use scheduling policies suitable for such systems. A scheduling policy must be applicable to large-scale systems involving large numbers of machines. Also, the policy must be fault-aware in the sense that it copes with resource volatility. Further adding to the complexity of scheduling for Desktop Grids is the inherent heterogeneity of such systems. Sub-optimal performance would result if the scheduling policy does not take into account information on heterogeneity. In this paper, we suggest and develop several scheduling policies for Desktop Grid systems involving different levels of heterogeneity. In particular, we propose a policy which utilizes the solution to a linear programming problem which maximizes system capacity. We consider parallel applications that consist of independent tasks.

1. Introduction

Widespread availability of low-cost, high performance computing hardware together with the rapid expansion of the Internet and advances in computing networking technology have led to an increasing use of heterogeneous computing (HC) systems. An HC system is constructed by networking various machines with different capabilities and coordinating their use to execute a set of tasks. Desktop Grids are HC systems characterized by the non-dedication of their machines. These systems aim to harvest a large number of desktop PCs owned by individuals and whose idle cycles can be exploited to run Grid applications. Desktop Grids have recently received a lot of attention because of the success of several popular applications such as SETI@home [19].

An important component of a Desktop Grid system is its scheduler. The scheduler is responsible for assigning resources to tasks. It uses a scheduling policy that is designed to optimize certain performance requirements. These scheduling policies may use certain information, such as task arrival rates and machine execution rates, to improve performance. Based on the information that can be used, scheduling policies are classified as static, dynamic, or adaptive (Shah *et al.* [20]). In a static policy, the scheduling is carried out independent of the state of the system and is done in a predetermined manner. A dynamic policy adapts its scheduling decisions based on the state of the system. Adaptive policies are dynamic policies where the parameters of the scheduling policy are changed based on the global state of the system. This paper suggests several adaptive scheduling policies for Desktop Grids.

A scheduling policy must support systems with a very large number of machines. Besides the natural complexity of scheduling for such large systems, the complexity is further complicated by several factors. First, Desktop Grids are characterized by very high resource volatility. In such systems, machines can fail at any time without any advance notice. Since Desktop Grids are typically based on the Internet, machines are also exposed to link failures. Furthermore, Desktop Grids are volunteer computing systems where participants voluntarily join in to execute the Grid applications. Thus, the machines of a Desktop Grid system are not dedicated (*i.e.,* machines' local jobs should have higher priority than the Grid tasks). To better cope with resource volatility, a scheduling policy must be fault-aware in the sense that it needs to exploit the knowledge of the effective computing power delivered by resources and the distribution of their fault times (if such information is available).

A second factor contributing to the complexity of scheduling for Desktop Grids is related to the heterogeneous nature of such systems. In this work, we consider heterogeneous machines which execute tasks that themselves may be highly heterogeneous. The execution time of a task depends on the class of the task as well as the executing

machines. Performance would be significantly impacted if information on task and machine heterogeneity is not taken into account by the scheduling policy. There is already work on developing policies for clusters of dedicated and heterogeneous machines (see Al-Azzoni and Down [1], He *et al.* [12], Maheswaran *et al.* [15], and [20]). To the best of our knowledge, this is the first paper to consider the problem of scheduling for heterogeneous Desktop Grids involving resource volatility.

Our workload model supports parallel applications consisting of independent tasks. These are used in a variety of domains, including simulations, fractal calculations, computational biology, and computer imaging. We assume that the Desktop Grid is mainly used to execute short-lived applications [13]. These applications consist of short tasks whose mean execution times are small relative to the mean machine availability times. Hence, for such applications, there is no need for incorporating fault tolerant scheduling mechanisms such as checkpointing, migration and replication.

In current Desktop Grids, the default scheduling policy is First-Come-First-Served (FCFS) [11, 13]. This policy does not require any information on task arrival rates and machine execution rates or availabilities. This policy performs well in systems with limited task heterogeneity. However, as our simulations show, the policy performance is suboptimal in systems with high task heterogeneity and degrades rapidly as the load increases. In this paper, we suggest the use of an existing policy (the Gcμ policy) which has been described in the queueing literature. This policy performs much better than the FCFS policy, but requires information on the machine execution rates. Furthermore, we develop a new policy (the LPAS_DG policy) which utilizes the solution to a linear programming (LP) problem that maximizes system capacity. In addition to the machine execution rates, this policy assumes knowledge of the task arrival rates and that there is a mechanism by which the scheduler detects machine failures and availabilities. Our simulation experiments show significant performance advantages for the LPAS_DG policy over the Gcμ policy, especially in highly heterogeneous systems.

The organization of the paper is as follows. Section 2 gives the workload model in detail and describes several Desktop Grid scheduling policies. The Gcμ policy and the LPAS_DG policy are described in Sections 2.3 and 2.4, respectively. In Section 3, we present the results obtained in our simulation experiments. The literature related to this work is discussed in Section 4. Section 5 concludes the paper and outlines future research work.

2. Fault-Aware Scheduling Policies for Desktop Grids

2.1. Workload Model

In our model for a Desktop Grid, there is a dedicated scheduler for assigning incoming tasks to the requesting machines. Let the number of machines in the system be M. It is assumed that the tasks are classified into 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.

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.* [7]).

Resource management systems for Desktop Grids mainly use pull-based scheduling (see Choi *et al.* [8, 9]). 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 [9], Domingues *et al.* [10], and [13]). Also, in pull-based scheduling, the scheduler makes a decision as soon as it receives a request from a machine [9].

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 scheduler becomes aware of the failure of any machine within a negligible amount of time [13]. We assume that the Desktop Grid is mainly used to execute short-lived applications [13]. 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. To model the non-dedication property of machines, we use an approach similar to [4]. 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. When a machine becomes available, it sends its request for a new task to the scheduler. As in [4], we assume that the ma-

chine 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). These estimates are obtained using techniques such as the ones suggested by Wolski *et al.* [21] and Yang *et al.* [22]. Thus, we can define the effective execution rate $\mu_{i,j}$ for the submitted tasks as follows:

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

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 $\mu_{i,j}$. As in [4, 13], once a task is submitted to a machine, the task can not be resubmitted unless a failure occurs.

2.2. Current Policies

A scheduling policy that is applicable to our workload model is the classical First-Come-First-Served (FCFS) policy. FCFS is used in major Desktop Grid schedulers [11, 13]. An advantage of FCFS is that it does not require any information about task arrival rates or machine execution rates. However, as our simulations show, FCFS only performs well in systems with limited task heterogeneity and under moderate system loads. As the application tasks become more heterogeneous and the load increases, performance degrades rapidly.

2.3. The Gcμ Policy

A related policy is a variation of the generalized cμ rule (Gcμ) analyzed by Mandelbaum and Stolyar [16]. We consider the version of the Gcμ rule which asymptotically minimizes delay costs. The policy can be stated as follows: when a machine j requests a task, the scheduler assigns it the longest waiting class i task such that $i \in \arg\max_i D_i(t)\mu'_{i,j}$, in which $D_i(t)$ is the longest sojourn time of a class i task at time t.

To the best of our knowledge, the Gcμ policy has never been suggested and used as a scheduling policy in Desktop Grids. The Gcμ policy aims at myopically maximizing the rate of decrease of the instantaneous delay cost. It has been proved that when the primitives α and μ satisfy certain conditions, the Gcμ policy minimizes both instantaneous and cumulative delay costs, asymptotically, over essentially all scheduling disciplines, preemptive or non-preemptive [16]. The optimality of the Gcμ policy is obtained under a heavy traffic assumption, in other words, optimality is achieved as the system load approaches 100 percent. When one backs off from the heavy traffic condition, we will see that there is room for making bad scheduling decisions, which in turn can significantly degrade performance.

Under moderate traffic conditions, the Gcμ rule could make more frequent bad scheduling decisions, especially in systems with highly heterogeneous execution rates. This results from the policy's greedy nature. Our LPAS_DG policy avoids this by preventing the assignment of particular task classes to inefficient machines.

Note that a scheduler using the Gcμ policy only requires information on the execution rates of the machines. Using this extra information, however, can result in achieving significant performance improvement over policies that do not use such information (*i.e.,* FCFS).

2.4. The LPAS_DG Policy

The Linear Programming Based Affinity Scheduling policy for Desktop Grids (LPAS_DG) 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.

$$\begin{aligned}
\max\ & \lambda \\
\text{s.t.}\ & \sum_{j=1}^{M} \delta_{i,j}\mu'_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N, && (1)\\
& \sum_{i=1}^{N} \delta_{i,j} \leq a_j, \qquad \text{for all } j = 1, \ldots, M, && (2)\\
& \delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \text{ and } j = 1, \ldots, M. && (3)
\end{aligned}$$

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 scheduler obtains values for a_j using the CPU availability predic-

tion techniques discussed in Section 4. Solving the allocation LP at each availability/non-availability event represents how the LPAS_DG policy adapts to the dynamics of machine availability. Constraint (2) enforces the condition that the allocation of machine j should not exceed its CPU availability. The use of a_j represents how the LPAS_DG policy adapts to the dynamics of CPU availability.

The value λ^* can be interpreted as follows. Consider an event in which a machine becomes available or unavailable. Let λ^* and $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, be an optimal solution to the allocation LP corresponding to the system just after the occurrence of the event. Consider the system that only consists of the available subset of the M machines. Then, the value λ^* can also be interpreted as the maximum capacity of this partial system [1, 12].

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 of making the scheduling decision t. 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 $\mu_{i,j}$ represents the effective execution rate for class i tasks at machine j ($\mu_{i,j} = a_j \mu'_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$). 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.

The allocation LP considers both the arrival rates and execution rates and their relative values in deciding the allocation of machines to tasks. In addition, these allocations are constrained by the CPU availabilities of the available machines. Consider a system with two machines and two classes of tasks ($M = 2$, $N = 2$). The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 1 & 1.5 \end{bmatrix} \text{ and } \mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}.$$

Assume that all machines are dedicated (*i.e.,* $a_j = 1$, for all $j = 1, \ldots, M$). Solving the allocation LP gives $\lambda^* = 1.764706$ and

$$\delta^* = \begin{bmatrix} 0 & 0.352941 \\ 1 & 0.647059 \end{bmatrix}.$$

Thus, when machine 1 requests a task, the scheduler only assigns it a class 2 task. Machine 2 can be assigned tasks belonging to any class. Although the fastest rate is for machine 1 at class 1, machine 1 is never assigned a class 1 task. Note that machine 1 is twice as fast as machine 2 on class 2 tasks and note that $\frac{\mu_{1,1}}{\mu_{2,1}} < \frac{\mu_{1,2}}{\mu_{2,2}}$.

There could be many optimal solutions to an allocation LP. These optimal solutions may have different number of zero elements in the δ^* matrix. The following proposition is a basic result in linear programming (the proof can be found in Andradóttir *et al.* [2]):

Proposition 1 *There exists an optimal solution to the allocation LP with at least $NM + 1 - N - M$ elements in the δ^* matrix equal to zero.*

Ideally, the number of zero elements in the δ^* matrix should be $NM + 1 - N - M$. If the number of zero elements is greater, the LPAS_DG policy would be significantly restricted in shifting workload between machines resulting in performance degradation. Also, if the number of zero elements is very small, the LPAS_DG policy resembles more closely the Gcμ policy. In fact, if the δ^* matrix contains no zeros at all, then the LPAS_DG policy reduces to the Gcμ policy. Throughout the paper, we use the unique optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros.

The LPAS_DG policy can be considered as an adaptive policy. As the policy only involves solving an LP, it is suited for scenarios when the global state of the system changes. For example, new machines can be added and/or removed from the system. Also, parameters such as the arrival rates and execution rates may change over time. On each of these events, one needs to simply solve a new LP and continue with the new values.

3. Simulation Results

We use simulation to compare the performance of the scheduling policies. In Section 3.1, we simulate an artificial system with high heterogeneity levels to show the impact of heterogeneity on performance. Then, in Section 3.2, we show the results of simulating a realistic Desktop Grid system.

The task arrivals are modeled by independent Poisson processes, each with rate α_i, $i = 1, \ldots, N$. The execution times are exponentially distributed with rates $\mu'_{i,j}$, where $1/\mu'_{i,j}$ represents the mean nominal execution time of a task of class i at machine j, $i = 1, \ldots, N$, $j = 1, \ldots, M$.

There are several performance metrics that can be used [4, 13]. We use the long-run average task completion time W, as a metric for performance comparison. A task completion time is defined as the time elapsing between the submission of the task and the completion of its execution, including resubmission times. For each simulation experiment, we also show the average task completion time for class i tasks, W_i, for all $i = 1, \ldots, N$.

Each simulation experiment models a particular system under different assumptions on machine and CPU availabilities. Each experiment is repeated 30 times. For every case,

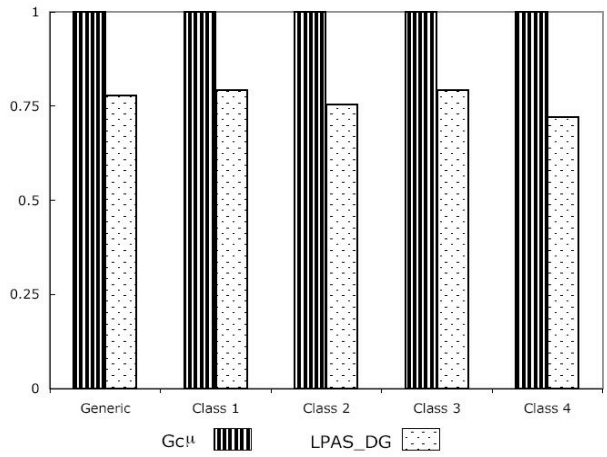

Figure 1. Relative average task completion times: System A under arrival rates α^1

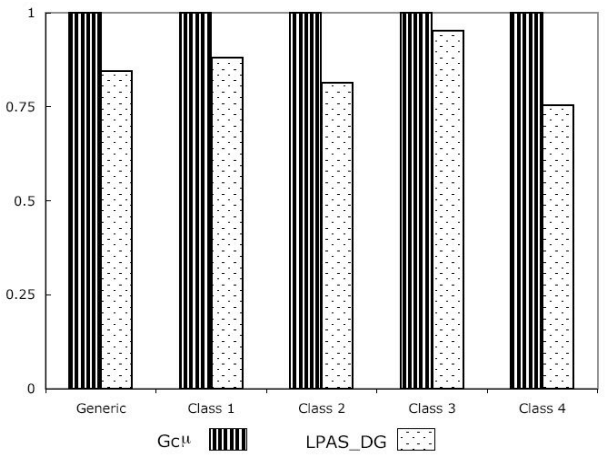

Figure 2. Relative average task completion times: System A under arrival rates α^2

we compute the 95%-confidence interval for W and W_i, $i = 1, \ldots, N$, with an accuracy of 0.1% or less. The accuracy of a confidence interval is calculated as the ratio of the half width of the interval over the mean value. We normalize the results with respect to the Gcμ policy. Note that we do not give performance results for the FCFS policy when it results in either an unstable system or one in which performance is several orders of magnitude worse.

3.1. Task and Machine Heterogeneity

There are different kinds of system heterogeneity. Machine heterogeneity refers to the average variation along the rows of μ, and similarly task heterogeneity refers to the average variation along the columns (see Armstrong [5]). In this section, we simulate a system with high task heterogeneity and high machine heterogeneity.

System A has $M = 28$ machines and $N = 4$ classes. The machines are partitioned into 7 groups (labeled G1 through G7). Each group consists of 4 machines and machines within a group are identical. The execution rates are shown as follows:

	Group						
Task	G1	G2	G3	G4	G5	G6	G7
1	4.5	2	9.5	6.2	10.25	2.25	3.95
2	6.2	4.5	6	2	4.2	5.9	10.25
3	9.5	6.5	4	10	5.9	2.25	3.95
4	2.25	10	2	3.95	1.75	10	1.75

Execution rates for System A

Figures 1 and 2 show simulation results for System A under two different arrival rates: $\alpha^1 = [50\ 48\ 50\ 48]$ and $\alpha^2 = [62.5\ 60\ 62.5\ 60]$. The arrival rates α^1 result in a lightly loaded system compared to a heavily loaded system under arrival rates α^2. We assume that each machine fails at the rate 0.05 (0.02) per time-unit and the mean fault time is four (two) time-units when modeling the system under arrival rates α^1 (α^2). Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$. It is assumed that machine fault times and availability times are exponentially distributed.

While the LPAS_DG policy achieves very competitive performance to that of the Gcμ policy, its performance is generally superior in highly heterogeneous and highly loaded systems (as the results above indicate). Furthermore, using the FCFS policy for System A results in instability *i.e.* the mean number of tasks in the system is unbounded. In general, the FCFS policy achieves poor performance and even results in unstable systems when there is high task heterogeneity and high machine heterogeneity. This suggests that FCFS will not be able to support the same level of throughput as our two proposed policies.

3.2. Realistic Architectures

To simulate more realistic scenarios, we use the data reported in [4, 6] which was collected by running benchmarking tools on an actual system. We refer to this system as System B.

In [4], the authors define the nominal computing power of a machine as a real number whose value is directly proportional to its speed. Thus, a machine with a nominal computing power of 2 is twice as fast as a machine with a nominal computing power of 1. It is found that, for System B, there are three different values for the nominal computing power of machines, namely $\{1, 1.125, 1.4375\}$.

Since we consider the problem of scheduling multiple applications on Desktop Grids, we define $P_{i,j}$ as the nomi-

nal computing power of machine j on class i tasks. Thus, a machine j with $P_{i,j} = 2$ is twice as fast as a machine j' with $P_{i,j'} = 2$ on class i tasks. In this manner, we can describe systems in which a machine is fast on some applications but slow on others.

As in [4], the CPU availability is described by a Markov chain whose parameters are computed using a network monitoring and forecasting system. A new value for the CPU availability is computed every 10 seconds of simulated time. The actual values for each machine's transition probabilities are reported in [6] (see Table 4.14). For the LPAS_DG policy, we compute a_j as the average CPU availability for each machine j from the corresponding Markov chain. This is justified for the model of System B since the mean execution time for a given task is much larger than the average time spent in a particular state of the Markov chain.

To model machine availability, we use a Weibull distribution. The actual values for the Weibull parameters depend on the particular machine. For System B, these parameters (shape and scale) are provided in Table 4.14 [6]. As in [4], the fault time of a machine is set to a constant 120 time-units.

We simulate two configurations based on System B ($B1$ and $B2$). Both systems consist of $M = 300$ machines. We group the machines into 15 groups. Each group consists of 20 machines identical in terms of the Markov chain describing CPU availability and the parameters for the Weibull distribution. Each group has the same parameters as those of one of the 15 machines of System B listed in Table 4.14 [6].

In System $B1$, we assume that the machines of a group are identical in terms of their nominal computing powers. Each group has the same nominal computing power as one of the 15 machines of System B. Furthermore, we assume that the nominal computing power of a machine depends only on the machine and is independent of the class of tasks being executed. Thus, if a machine j belongs to a group G and the nominal computing power for the group is P_G, then $P_{i,j} = P_G$, for all $i = 1, \ldots, N$. Thus, a fast machine is fast on all applications. System $B1$ represents a system which is mainly used to execute a single application.

In System $B2$, we assume that each machine has a nominal computing power (on class i tasks) $P_{i,j}$ randomly chosen from $\{1, 1.125, 1.4375\}$ with equal probabilities. Thus, a machine can be fast executing some applications while, at the same time, slow executing other applications. System $B2$ represents a system which is mainly used to execute multiple applications with inherent heterogeneity.

Finally, we assume that there are $N = 4$ classes (or applications). The authors in [4] define $BaseTime$ as the mean execution time of a task submitted to a machine with a nominal computing power of 1. Thus, each class consists of tasks with the same value for $BaseTime$ (for class i, we denote it by $BaseTime_i$). We assume that $BaseTime_i =$ 8750, 17500, 35000, 50000, for $i = 1, \ldots, 4$, respectively. This information is enough to generate the matrix μ'. The mean nominal execution time for a class i task at machine j can be computed as $BaseTime_i * 1/P_{i,j}$.

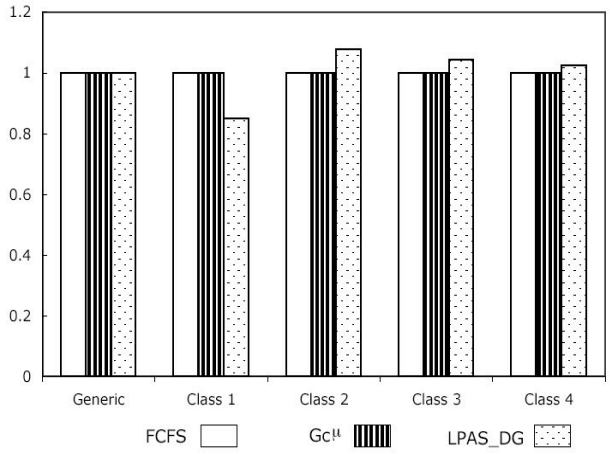

Figure 3. Relative average task completion times: System $B1$ under arrival rates α^3

Figures 3 and 4 show simulation results for Systems $B1$ and $B2$ under arrival rates $\alpha^3 = [0.00457\ 0.00229\ 0.00114\ 0.0008]$. These results indicate that the FCFS policy achieves acceptable performance in systems with low task heterogeneity, such as System $B1$. However, as the level of task heterogeneity increases (*e.g.* System $B2$), FCFS results in performance degradation which gets worse as the load increases. For instance, Figure 5 shows results for System $B2$ under higher load ($\alpha^4 = [0.00495\ 0.0011\ 0.00214\ 0.00135]$). For such cases, both the Gcμ and the LPAS_DG policies result in significant performance improvement. The LPAS_DG policy is generally superior in highly heterogeneous and highly loaded systems.

4. Literature Review

A taxonomy of Desktop Grids and a survey focusing on scheduling is provided in [9]. This taxonomy is defined by three major components: the application's perspective, the resource provider's perspective, and the scheduler's perspective. With respect to our workload model, we consider applications with independent, fixed tasks that are computation-intensive. There are no deadlines associated with tasks and the tasks arrive non-deterministically to the scheduler. In terms of the resource provider's perspective, we assume that the resource providers (*i.e.*, the machines) are not dedicated to public execution and they are faulty. In terms of the scheduler's perspective, a centralized organization is assumed. The scheduler uses pull-based schedul-

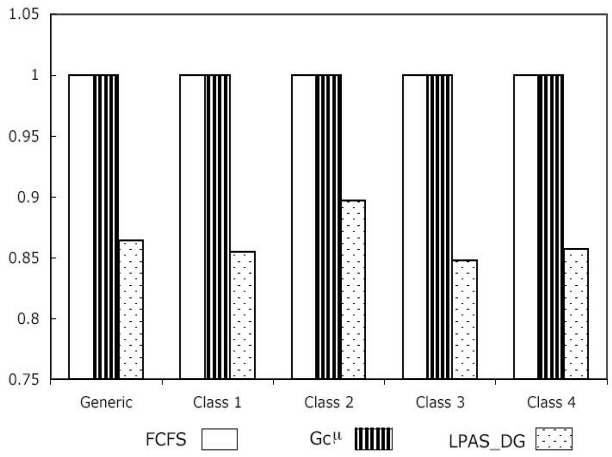

Figure 4. Relative average task completion times: System $B2$ under arrival rates α^3

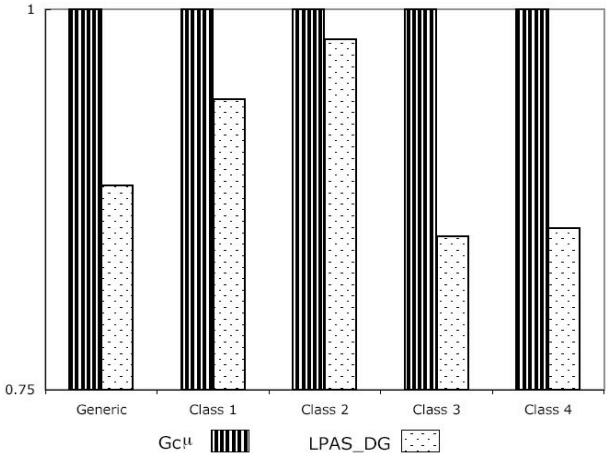

Figure 5. Relative average task completion times: System $B2$ under arrival rates α^4

ing in which scheduling events are initiated by the resource providers.

Several fault-aware Desktop Grid scheduling policies are presented in [4] for Bag-of-Tasks applications. The proposed policies exploit fault handling mechanisms including replication and checkpointing. Furthermore, these policies exploit knowledge of the effective computing power delivered by resources and the distribution of their fault times to improve scheduling performance. The performance of the different policies is analyzed using an extensive simulation study. The policies proposed in [4] assume that the set of tasks is initially available to the scheduler, however, we assume a continuous arrival stream of tasks and that the scheduler only knows the arrival rates and execution rates (does not need to know the entire distribution). Our work goes beyond this by addressing workloads where multiple Bag-of-Tasks applications are simultaneously submitted.

Other heuristics are proposed in [13]. These heuristics attempt to minimize the overall execution time, or the makespan, of a single parallel application. The application is assumed to consist of a number of independent tasks that is relatively small compared to the number of available resources. The heuristics are based on three resource selection techniques, namely resource prioritization, resource exclusion, and task replication. Even though the heuristics developed in [13] are designed to schedule a single application, the authors acknowledge that these heuristics provide key elements for designing effective "job scheduling" strategies. Furthermore, the authors planned to design scheduling heuristics for the scenario where multiple applications are submitted over time. In this context, our work represents a step in addressing such environments.

Several scheduling techniques are suggested in [10] for institutional Desktop Grids. Institutional Desktop Grids are grids comprised of the desktop machines of an institution (academic or corporate) and thus characterized by a more homogenous computing infrastructure. Similar to [4, 13], the scheduling techniques are designed to minimize the turnaround time of a single Bag-of-Tasks application. The turnaround time for a Bag-of-Tasks application is defined as the elapsed time between the submission of the first task until the last task is completed.

Several papers study machine availability in Desktop Grids. In Nurmi *et al.* [17], availability data is collected from different Desktop Grid environments. Their results indicate that either a hyperexponential or Weibull distribution effectively represents machine availability in enterprise and Internet computing environments. In Kondo *et al.* [14], statistics from four real enterprise Desktop Grids are gathered in order to develop predictive models for machine availability.

An approach for predicting machine availability in Desktop Grids is presented in Ren *et al.* [18]. The authors apply semi-Markov process models for the prediction. Their experimental results show that the prediction has an accuracy of 86% on average and it is robust. They suggest a method for applying availability prediction to job scheduling. Using simulation, they show the effectiveness of their scheduling policies in large compute-bound guest applications. Our work proposes policies for short-lived applications.

A significant amount of work has been done on the measurement and characterization of CPU availability. The work of [22] includes techniques based on time series predictors for predicting CPU load at some future time point, average CPU load for some future time interval, and variation of CPU load over some future time interval. The ex-

pected future variance in CPU availability is used to derive a conservative scheduling policy to make data mapping decisions for a particular class of applications, namely, loosely synchronous, iterative, data-parallel computations. The work of [21] examines the problem of making short and medium term forecasts of CPU availability on time-shared Unix systems. Their results demonstrate the possibility of making short and medium term predictions of available CPU performance despite the presence of long-range autocorrelation and potential self-similarity.

5. Conclusion

In this paper, we have proposed to use the Gcμ policy for Desktop Grids when information on the machine execution rates are available. When task arrival rates and CPU availabilities are available, we have developed the LPAS_DG policy which utilizes the solution to an allocation LP. Both policies perform much better than FCFS, especially for applications with high task heterogeneity. A distinct feature for this work is the proposal of fault-aware policies that take into consideration the heterogeneity of Desktop Grids.

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, to appear.

[2] S. Andradóttir, H. Ayhan, and D. G. Down. Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, 51(6):952–968, 2003.

[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] M. Canonico. Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids. PhD thesis, University of Turin, 2006.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] Y.-T. He, I. Al-Azzoni, and D. Down. MARO - MinDrift affinity routing for resource management in heterogeneous computing systems. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, pages 71–85, 2007.

[13] 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.

[14] 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.

[15] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 30–44, 1999.

[16] A. Mandelbaum and A. L. Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized cμ-rule. *Operations Research*, 52(6):836–855, 2004.

[17] 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.

[18] X. Ren, S. Lee, R. Eigenmann, and S. Bagchi. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, 5(2):173–195, 2007.

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

[20] R. Shah, B. Veeravalli, and M. Misra. On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1675–1686, 2007.

[21] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared Unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.

[22] L. Yang, J. M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the ACM/IEEE conference on Supercomputing*, page 31, 2003.

Hostname	P	shape	scale	P_{aa}	P_{ab}	P_{ac}	P_{ba}	P_{bb}	P_{bc}	P_{ca}	P_{cb}	P_{cc}
bird	1	0.568664	477343	0.998	0.001	0.001	0.015	0.970	0.015	0.000	0.0197	0.9803
blind	1.125	0.729215	856663	0	0	0	0	0.9814	0.0186	0	0.0098	0.9902
booboo	1	0.570816	619497	0	0	0	0	0.9904	0.0096	0	0.0133	0.9867
chocolate	1.4375	0.662438	610445	0	0	0	0	0.9958	0.0042	0	0.1376	0.8624
hobbes	1.125	0.560362	199690	0.9969	0.0029	0.0002	0.0270	0.9674	0.0056	0.0005	0.0123	0.9872
joplin	1	0.969769	1271536	0.9987	0.0011	0.00003	0.0028	0.9946	0.0024	0.0002	0.0134	0.9863
kenny	1.125	0.729823	350024	0	0	0	0	0.9914	0.0086	0	0.0152	0.9848
marge	1.4375	0.677637	373307	0.9982	0.0017	0.0001	0.0097	0.9735	0.0168	0.0005	0.0449	0.9546
marvin	1	0.928094	753368	0	0	0	0	0.9795	0.0205	0	0.0378	0.9622
miles	1.125	0.570816	619497	0	0	0	0	0.9933	0.0067	0	0.0503	0.9497
nat	1.4375	0.607016	405233	0	0	0	0	0.9946	0.0054	0	0.0352	0.9648
popeye	1	0.616905	228117	0	0	0	0	0.9889	0.0111	0	0.0214	0.9786
rocky	1.125	0.537631	178959	0	0	0	0	0.9964	0.0036	0	0.0004	0.9996
scooby	1.4375	0.68684	248058	0.9982	0.0016	0.0002	0.0093	0.9815	0.0092	0.0005	0.0169	0.9826
taz	1.4375	0.556867	243961	0	0	0	0	0.9916	0.0084	0	0.025	0.9750

Table 4.14: CSIL resources

Figure 6. (for Reviewers Use Only) A copy of Table 4.14 in [6] which is used in Section 3.2 to provide the parameters needed to simulate System B