Decentralized Load Balancing for Heterogeneous 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*—Several decentralized load balancing policies have been proposed to address the issue of scalability in grids. However, the communication overhead incurred in exchanging state information remains a burden. In this paper, we propose a dynamic, decentralized load balancing policy which performs very competitively in heterogeneous grids. The policy uses an effective mechanism for state information exchange, which significantly reduces the communication overhead, while quickly updating the state information in a decentralized fashion.**

***Keywords*-grid computing; decentralized load balancing; distributed systems; heterogeneous processors; information exchange;**

I. Introduction

Widespread availability of low-cost, high performance computing hardware and the rapid expansion of the Internet and advances in computing networking technology have led to an increasing use of heterogeneous computing (HC) systems (see Kontothanassis and Goddeau [1]). An HC system is constructed by networking various machines with different capabilities and coordinating their use to execute a set of tasks. Grids are an example of HC systems.

To harness the computational power of a grid, a load balancing policy is used. Such policies attempt to balance the load with the end result of maximizing resource utilization and hence optimizing performance. Load balancing for grids is complicated due to several factors. Grids can consist of a very large number of machines. The state of the system dynamically changes and the load balancing policy should adapt its decisions to the state of the system. Another factor contributing to the complexity of load balancing for grids is related to the heterogeneous nature of such systems. These systems interconnect a multitude of heterogeneous machines (desktops with various resources: CPU, memory, disk, etc.) to perform computationally intensive applications that have diverse computational requirements. Performance may be significantly impacted if information on task and machine heterogeneity is not taken into account by the load balancing policy.

In this paper, we consider decentralized load balancing policies. In centralized policies, a central machine is dedicated as a load balancer and tasks are submitted to the central machine. Thus, the load balancer becomes a bottleneck and a single point of failure. To avoid this, decentralized load balancing policies involve all machines in load balancing and avoid the use of a central server.

Even though decentralized load balancing has advantages in terms of scalability and fault tolerance, the communication overhead incurred by frequent state information exchange between machines represents a challenge. In such systems, the state information needs to be propagated to each machine. In our earlier work [2], we have developed a centralized load balancing policy, the Linear Programming based Affinity Scheduling policy (LPAS), that performs competitively in heterogeneous systems. The policy uses the solution to an allocation linear programming problem (LP) which maximizes the system capacity.

Adapting the policy into a decentralized setting is not straightforward (the difficulties are discussed in Section IV). It requires all machines to know the up-to-date state of other machines. The communication overhead necessary to do this looks initially prohibitive. In this work, we propose an effective mechanism for state information update designed for the LPAS policy. The mechanism significantly cuts down the communication overhead while quickly updating the state information. Our simulations show significant performance advantages over competing policies, especially in highly heterogeneous systems. Although this paper provides a proof of concept, future work is needed to deploy the proposed policy on actual grids in order to analyze performance under representative workloads.

The organization of the paper is as follows. Section II gives the workload model in detail. Section III describes several decentralized load balancing policies and gives an overview of the related work. The LPAS_dec policy is described in Section IV. In Section V, we present the results obtained in our simulation experiments.

II. Workload Model

In decentralized load balancing, a task can be submitted to any machine in the grid. Each machine is responsible for the assignment of its locally submitted tasks to one of the grid machines. Upon the arrival of a task to a machine, the machine immediately makes a decision on whether to execute the task locally or send it for execution on another machine. After that, the task can only be executed by the machine to which it is assigned. Thus, a task can not be migrated more than once. Similarly, several researchers assume a migration limit of one as task migration is often difficult in practice and there are no significant benefits of using higher migration limits (see Lu *et al.* [3] and Shah *et al.* [4]).

We consider dynamic load balancing policies. These policies, as opposed to static policies, attempt to exploit dynamic state information to optimize performance. In order to do that, certain types of information need to be exchanged among the machines, *e.g.* task queue lengths, machine execution rates, and so forth. In grids, there is no efficient state-broadcast mechanism [3]. Other approaches for information exchange, such as state-polling, are also problematic in practice (see [3], Gu *et al.* [5], and Werstein *et al.* [6]). To minimize the overhead of information collection, we assume that state information exchange is done by mutual information feedback [3]. Thus, when a machine j_1 needs the local state information of another machine j_2, then it sends a request message to j_2 which in turn sends back a reply message. Both the request and reply messages may embed other local state information as dictated by the load balancing policy.

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.* [7]) or parameter-sweep applications (as in Casanova *et al.* [8]). Such applications are becoming predominant for grids (see Iosup *et al.* [9] and Li and Buyya [10]).

While determining the exact task execution time on a target machine remains a challenge, there exists several techniques that can be used to estimate an expected value for the task execution time (see Rao and Huh [11]). The policies considered in this paper exploit estimates on mean task execution times rather than exact execution times. Furthermore, in grids, tasks that belong to the same application are typically similar in their resource requirements. For example, some applications are CPU bound while others are I/O bound. In fact, several authors have observed the high dependence of a task execution time on the application it belongs to and the machine it is running on. They argue for using application profile information to guide resource management (see [1]). We follow the same steps and assume that the tasks are classified into groups (or classes) with identical distributions for the execution times.

In our model for a grid, it is assumed that the tasks are classified into N classes. Tasks of class i arrive to a machine j at the rate $\alpha_{i,j}$. Let α be the arrival rate matrix, having (i, j) entry $\alpha_{i,j}$. Let the number of machines in the system be M. Thus, $\alpha_i = \sum_{j=1}^{M} \alpha_{i,j}$ is the total arrival rate of class i tasks to the system. Let $\mu_{i,j}$ be the execution rate for tasks of class i at machine j, hence $1/\mu_{i,j}$ is the mean execution time for class i tasks at machine j. We allow $\mu_{i,j} = 0$, which implies machine j is physically incapable of executing class i tasks. Each task class can be executed by at least one machine. Let μ be the execution rate matrix, having (i, j) entry $\mu_{i,j}$.

We note that performance monitoring tools such as NWS [12] and MonALISA [13] can be used to provide dynamic information on the state of the grid system. Furthermore, these tools anticipate the future performance behaviour of an application including task arrival and machine execution rates.

III. Current Policies and Related Work

The MCT (minimum completion time) policy assigns an arriving task to the machine that has the earliest expected completion time. Several authors have suggested decentralized load balancing policies that are based on the MCT policy, *e.g.*, the IDP (Instantaneous Distribution Policy) in [3] and the LBA (Load Balancing on Arrival) policy in [4]. When a task arrives to a machine, the machine contacts all machines in the system to determine the machine with the earliest expected completion time.

The policy suffers from a significant information exchange overhead. In particular, for each arriving task to a machine, the machine needs to send a request message to all machines in the system who in turn need to send back a reply message containing the expected completion time information. Thus, a total of $2 \times (M - 1)$ message exchanges are needed for every arriving task.

An advantage of the MCT policy is that a machine does not require any information about the task arrival rates or machine execution rates of other machines. Thus, only the expected completion times need to be exchanged between machines. It is for this reason that the MCT policy is suited for systems where predicting these rates is not possible or severely inaccurate.

In order to address the limitations of the MCT policy, the k-percent best (KPB) policy (Maheswaran *et al.* [14]) identifies for each class a subset consisting of the $(kM/100)$ best machines based on the execution times for the class, where $100/M \leq k \leq 100$. With respect to a machine j, let $S_i^{k,j}$ be the set of the $\lfloor kM/100 \rfloor$ machines that have the smallest expected execution time for class i tasks. When a task of class i arrives to machine j, the machine assigns the task to the machine in the subset $S_i^{k,j}$ that has the earliest expected completion time. Define $\overline{k} = \lfloor kM/100 \rfloor$ to be the number of machines considered by the KPB policy.

Depending on the value of $\overline{k}$, an advantage of the KPB policy is that it requires less state information than the MCT policy. Furthermore, this can significantly reduce the number of message exchanges. In fact, a total of $2 \times \overline{k}$ message exchanges are needed for every arriving task. This can be a dramatic improvement over the MCT policy.

While the KPB policy succeeds in reducing the required state information, setting its parameter (k) may be problematic and can only be done in an ad-hoc manner. Instability or severe performance degradation can result if inappropriate values for k are used. Also, the KPB policy maps each class to the same number of machines, which may not be desirable.

The KPB policy requires knowledge on machine execution rates while the MCT policy does not. We use the following mechanism for exchanging information on machine execution rates. When a task of class i arrives to a machine j', the machine sends request messages to the machines $j \in S_i^{k,j'}$. In each request message, machine j' includes its local execution rates ($\mu_{i,j'}$, $i = 1, \ldots, N$). Upon receiving the request messages, each of the contacted machines replies with a message including the corresponding expected completion time as well as the local execution rates. Thus, at the end, machine j' and the machines $j \in S_i^{k,j'}$ update their local state information with the corresponding execution rates.

The MCT and KPB policies balance the load only upon task arrivals, as do the LPAS_dec policy, [4], [11], and Arora *et al.* [15]. Other policies [6] are threshold-based *e.g.*, only when the load on a machine exceeds a certain threshold, load balancing is triggered. Some load balancing policies use a combination of both techniques (see [3] and [5]).

On the event of load balancing, a policy should determine which tasks to migrate. Policies such as the LPAS_dec policy, which balance the load upon task arrivals, typically migrate only the arriving task. Some policies, however, migrate additional tasks, such as [15]. Other policies rank the queued tasks based on certain criteria and only migrate the highest ranking tasks (see [3] and [5]).

Two mechanisms for information exchange were discussed in Section II: state polling and mutual information feedback. Several load balancing policies that use state polling are presented in [4], [5], and [6]. The LPAS_dec policy uses mutual information feedback, as do the policies presented in [3], [11], and [15].

IV. The LPAS_dec Policy

The LPAS_dec policy is similar to the KPB policy in that only a subset of machines need to be considered for each class, however, the determination of this subset requires solving the following LP (Andradóttir *et al.* [16]), where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$.

$$\max \ \lambda$$

$$\text{s.t.} \sum_{j=1}^{M} \delta_{i,j}\mu_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N, \tag{1}$$

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

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

The interpretation of the variables and constraints is identical to that of the allocation LP in [2].

Let λ^* and $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, be an optimal solution to the allocation LP. Let δ^* be the machine allocation matrix where the (i,j) entry is $\delta^*_{i,j}$.

The LPAS_dec policy can be stated as follows. Each machine j' solves a local version (using local data) of the allocation LP to find $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. When a new task of class i arrives to a machine j', let $S_i^{j'}$ denote the set of machines whose corresponding $\delta^*_{i,j}$ at machine j' is not zero. Machine j' assigns the task to the machine $j \in S_i^{j'}$ that has the earliest expected completion time among the subset of machines $S_i^{j'}$. Again, ties are broken arbitrarily.

The LPAS_dec policy requires knowledge on both arrival and execution rates. We use the following mechanism for information exchange. When a task of class i arrives to a machine j', the machine sends request messages to the machines $j \in S_i^{j'}$. In each request message, machine j' includes its local arrival and execution rates ($\alpha_{i,j'}$, $\mu_{i,j'}$, $i = 1, \ldots, N$). Upon receiving the request messages, each of the contacted machines replies with a message including the corresponding expected completion time as well as the local arrival and execution rates. Thus, at the end, machine j' and the machines $j \in S_i^{j'}$ update their local state information with the corresponding arrival and execution rates.

In the ideal conditions when full state information is available, all the machines solve the same allocation LP and thus use the same δ^* matrix which is necessary to achieve the maximum capacity. In such cases, our earlier work in [2] shows that the LPAS_dec policy performs well in heterogeneous systems. However, in practice, at any given time, one or more of the machines may have different views of the state of the system. Thus, they solve different allocation LPs and the resulting δ^* matrices are different. However, as our simulation experiments indicate, the information exchange mechanism of the LPAS_dec policy is effective in its state update and thus the machines tend to quickly have the same view of the system. Furthermore, since the LPAS_dec policy does not use the actual values for $\{\delta^*\}$ (it only uses information on what entries are nonzero), and since these LPs are inherently robust, the resulting δ^* matrices tend to be similar with respect to the positions of the zero and nonzero entries. Thus, performance would not be significantly deteriorated when the observed system state is a little different amongst the machines.

Consider a system with two machines and two classes of tasks ($M = 2$, $N = 2$). Assume initially that α and μ are known by both machines:

$$\alpha = \begin{bmatrix} 1 & 1.45 \\ 1 & 1.45 \end{bmatrix}, \text{ and } \mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}.$$

Solving the allocation LP gives

$$\delta^* = \begin{bmatrix} 0 & 0.5 \\ 1 & 0.5 \end{bmatrix}.$$

Thus, all arriving tasks that belong to class 1 are assigned to machine 2. At the times of their arrivals, tasks that belong to class 2 are assigned to the machine, either machine 1 or 2, that has the earliest expected completion time. Even though machine 1 has the fastest rate for class 1, class 1 tasks are never assigned to it. Since the system is highly loaded, and since $\frac{\mu_{1,1}}{\mu_{2,1}} < \frac{\mu_{1,2}}{\mu_{2,2}}$ and $\alpha_1 = \alpha_2$, the performance is improved significantly if machine 1 only executes class 2 tasks.

Now, assume that $\alpha_{1,2}$ becomes 0.5. Thus, machine 2 solves a new allocation LP to obtain:

$$\delta^* = \begin{bmatrix} 0 & 0.3273 \\ 1 & 0.6727 \end{bmatrix}.$$

Even though machines 1 and 2 use different δ^* matrices until the next state update, they are equivalent in terms of the locations of the zero and nonzero entries. Thus, machine 1 still uses an allocation matrix that is equivalent in effect to the allocation matrix which maximizes the system capacity.

Consider another scenario in which $\mu_{1,2}$ becomes 0.5. Thus, machine 2 solves a new allocation LP to obtain:

$$\delta^* = \begin{bmatrix} 0.2727 & 0 \\ 0.7273 & 1 \end{bmatrix}.$$

Thus, machines 1 and 2 use different δ^* matrices (even with respect to the locations of the zero and nonzero entries). However, one can check that, via the information exchange mechanism of the LPAS_dec policy, machine 1 becomes aware of the change of $\mu_{1,2}$ upon the arrival of a task to either machine. Hence, machine 1 is quickly updated with the new state of machine 2 and thus both machines solve the same allocation LP. In larger systems, there will be a delay in propagating information on the state change throughout the system. However, our simulation experiments suggest that the state update happens quickly enough to not significantly impact performance.

In the LPAS_dec policy, tasks of each class can only be assigned to a subset of machines. Ideally, the size of each subset should be much smaller than M. To achieve this, the δ^* matrix should contain a large number of elements that are equal to zero. In fact, there could be many optimal solutions to an allocation LP, and an optimal solution with a larger number of zeros in the δ^* matrix is preferred. The following proposition gives the number of zero elements in the δ^* matrix that could be achieved (the proof can be found in [16]):

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_dec policy would be significantly restricted in shifting workload between machines resulting in performance degradation. Furthermore, in this case, our information exchange mechanism becomes less effective in its state update. Also, solutions that result in degenerate cases should be avoided. For example, if the δ^* matrix contains no zeros at all, then the LPAS_dec policy reduces to the MCT policy. Throughout the paper, we use an optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros.

V. Simulation Results

We use simulation to compare the performance of several load balancing policies including the LPAS_dec policy. In Section V-A, we simulate an artificial system with high heterogeneity levels to show the impact of heterogeneity on performance. Then, in Section V-B, we show the results of simulating a realistic grid.

The task arrivals are modeled by independent Poisson processes, each with rate $\alpha_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. The execution times are exponentially distributed with rates $\mu_{i,j}$, where $1/\mu_{i,j}$ represents the mean 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. We use the long-run average task completion time W, as a metric for performance comparison. A task completion time is defined as the elapsed time between the submission of the task and the completion of its execution. For each simulation experiment, we also show the average task completion time for class i tasks, W_i, for all $i = 1, \ldots, N$. Another metric we also show is the total number of message exchanges, X. With respect to a given policy, a larger value for X indicates more overhead is involved in state information exchange.

In this section, we define several systems. Each simulation experiment models a particular system, characterized by the values of M, N, $\alpha_{i,j}$, and $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Each experiment is repeated 30 times. For every case, we give W, W_i, $i = 1, \ldots, N$, and X. For W, we also give the accuracy of the confidence interval defined as the ratio of the half width of the interval over the mean value (all statistics are at 95% confidence level).

A. 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 [17]). In this section, we simulate a system with high task heterogeneity and high machine heterogeneity.

System A has $M = 7$ machines and $N = 4$ classes. Define

Table I
SIMULATION RESULTS FOR SYSTEM A

Policy	W	W_1	W_2	W_3	$W4$	Δ
MCT	**3.41** $\pm 13.31\%$	3.40	3.40	3.40	3.43	**0%**
KPB $\overline{K}=2$	**0.37** $\pm 4.04\%$	0.28	0.58	0.40	0.20	**66.67%**
KPB $\overline{K}=3$	**0.24** $\pm 0.59\%$	0.20	0.25	0.27	0.26	**50.02%**
LPAS_ dec	**0.22** $\pm 0.36\%$	0.23	0.19	0.24	0.21	**65.19%**

α' and μ' as follows:

$$\alpha' = \begin{bmatrix} 2 & 1.5 & 1.75 & 1 & 3 & 1.9 & 1.35 \\ 1.35 & 1.5 & 2.4 & 1.55 & 2.9 & 0.75 & 1.55 \\ 4 & 2.75 & 1 & 1.35 & 1.5 & 0.9 & 1 \\ 2 & 1.75 & 2 & 1.5 & 2.25 & 1.75 & 0.75 \end{bmatrix}$$

and

$$\mu' = \begin{bmatrix} 4.5 & 2 & 9.5 & 6.2 & 10.25 & 2.25 & 3.95 \\ 6.2 & 4.5 & 6 & 2 & 4.2 & 5.9 & 10.25 \\ 9.5 & 6.5 & 4 & 10 & 5.9 & 2.25 & 3.95 \\ 2.25 & 10 & 2 & 3.95 & 1.75 & 10 & 1.75 \end{bmatrix}.$$

Initially, the arrival and execution rates are given by $\alpha = \alpha'$ and $\mu = \mu'$. The rates only change at regular rate-change events. At every rate-change event, only a single rate from $\alpha_{i,j}$ or $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, changes randomly with equal probabilities. Time intervals between the rate-change events are exponentially distributed with mean $1/0.035$ time-units. For a change in α, $\alpha_{i,j}$ is set to $\alpha'_{i,j}$, $1.1\alpha'_{i,j}$, or $1.2\alpha'_{i,j}$ with equal probabilities. For a change in μ, $\mu_{i,j}$ is set to $\mu'_{i,j}$, $1.05\mu'_{i,j}$, or $1.15\mu'_{i,j}$ with equal probabilities. Thus, the system experiences different loads with the lowest average load of 77.59% (when $\alpha = \alpha'$ and $\mu = 1.15\mu'$) and the highest average load of 89.23% (when $\alpha = 1.2\alpha'$ and $\mu = \mu'$).

Table I shows simulation results for System A. We simulate the execution of the system for 200,000 time-units. In the last column of the table, we define Δ as the improvement in the total number of message exchanges (X) with respect to the MCT policy. For X, the accuracy of the confidence intervals is less than 0.1%.

The MCT policy performs much worse than the other policies. In general, the MCT policy achieves poor performance and even results in unstable systems when the system is highly loaded and there is high task heterogeneity and high machine heterogeneity. Using the KPB policy, performance is dramatically improved with respect to the MCT policy. However, finding an appropriate value for $\overline{k}$ is problematic. Furthermore, depending on the value of $\overline{k}$, there is a tradeoff between the achieved performance (W) and the overhead of message exchanges (X). The LPAS_dec policy achieves the best performance for System A even though there is a dramatic decrease in the total number of message exchanges. It results in values for Δ comparable with those of the KPB policy with $\overline{K} = 2$, while achieving an improvement of 40% in the average task completion time.

B. Realistic Architectures

To simulate more realistic scenarios, we use the data reported in [7] and Canonico [18] which was collected by running benchmarking tools on an actual system.

In [7], 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 the monitored system, there are three different values for the nominal computing power of machines, namely $\{1, 1.125, 1.4375\}$.

Since our load balancing policies consider multiple applications on grids, we define $P_{i,j}$ as the nominal 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 $P_{i,j'} = 1$ on class i tasks. In this manner, we can describe systems in which a machine is fast on some applications but slow on others.

To model varying machine execution rates, we use an approach similar to [7]. When a machine is available, it may also run local jobs (*i.e.*, jobs submitted by a local user). When a machine is busy with local jobs, the result is a slowing down of the execution of the grid tasks. To model the non-dedication property of a machine j, we define its CPU availability (a_j) as the expected proportion of time that it is going to spend in executing the grid tasks. Thus, the execution rate for class i tasks at machine j is effectively $\mu_{i,j} \times a_j$. We assume that the load balancing policies use these estimated effective execution rates.

As in [7], 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 [18] (see Table 4.14). For the LPAS_dec 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 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.

We define the following system. System B consists of $M = 300$ machines which are grouped into 15 groups. We simulate the execution of the system for two billion time-units. To have dynamic CPU availabilities, the CPU availability for each machine changes randomly at regular rate-change events (time intervals between such events are exponentially distributed with mean 5 million time-units). At every rate-change event, each machine assumes the same

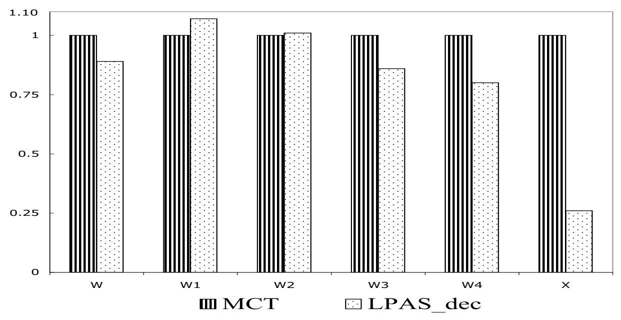

Figure 1. Relative average task completion times and number of message exchanges: System B under arrival rates α

parameters as one of the 15 machines listed in Table 4.14 in [18] with equal probabilities.

In System B, 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 B 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 [7] 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. Assuming $a_j = 1$, the mean execution time for a class i task at machine j can be computed as $BaseTime_i \times 1/P_{i,j}$.

Figure 1 show simulation results for System B under arrival rates $\alpha = [0.00457 \;\; 0.00229 \; 0.00114 \; 0.00080]$. For a machine j, we assume that $\alpha_{i,j} = \alpha_i/M$, $i = 1, \ldots, N$. In this section, we normalize the results with respect to the MCT policy and note that the accuracy of the generated confidence intervals is 0.1% or less. The KPB policy is not included as it is difficult to find an optimal value for $\overline{k}$. As the results indicate, the LPAS_dec policy achieves better performance than the MCT policy with the added advantage of significant reduction in the number of message exchanges.

References

[1] L. Kontothanassis and D. Goddeau, "Profile driven scheduling for a heterogeneous server cluster," in *Proceedings of the 34th International Conference on Parallel Processing Workshops*, 2005, pp. 336–345.

[2] I. Al-Azzoni and D. G. Down, "Linear programming based affinity scheduling of independent tasks on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, pp. 1671–1682, 2008.

[3] K. Lu, R. Subrata, and A. Y. Zomaya, "On the performance-driven load distribution for heterogeneous computational grids," *Journal of Computer and System Sciences*, vol. 73, no. 8, pp. 1191–1206, 2007.

[4] 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*, vol. 18, no. 12, pp. 1675–1686, 2007.

[5] D. Gu, L. Yang, and L. R. Welch, "A predictive, decentralized load balancing approach," in *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, 2005.

[6] P. Werstein, H. Situ, and Z. Huang, "Load balancing in a cluster computer," in *Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2006, pp. 569–577.

[7] 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*, 2006, pp. 56–63.

[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*, 2000, pp. 349–363.

[9] 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*, 2008, pp. 97–108.

[10] H. Li and R. Buyya, "Model-driven simulation of grid scheduling strategies," in *Proceedings of the 3rd International Conference on e-Science and Grid Computing*, 2007, pp. 287–294.

[11] I. Rao and E.-N. Huh, "A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing," *Journal of Supercomputing*, vol. 45, no. 2, pp. 185–204, 2008.

[12] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 757–768, 1999.

[13] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre, "MonALISA: an agent based, dynamic service system to monitor, control and optimize grid based applications," in *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, 2004.

[14] 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*, 1999, pp. 30–44.

[15] M. Arora, S. K. Das, and R. Biswas, "A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments," in *Proceedings of the International Conference on Parallel Processing Workshops*, 2002, pp. 499–505.

[16] S. Andradóttir, H. Ayhan, and D. G. Down, "Dynamic server allocation for queueing networks with flexible servers," *Operations Research*, vol. 51, no. 6, pp. 952–968, 2003.

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

[18] M. Canonico, "Scheduling algorithms for Bag-of-Tasks applications on fault-prone desktop grids," Ph.D. dissertation, University of Turin, 2006.