Linear Programming Based Affinity Scheduling for Heterogeneous Computing Systems

Issam Al-Azzoni
Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

Abstract *Resource management systems (RMS) are an important component in heterogeneous computing (HC) systems. One of the jobs of an RMS is the mapping of arriving tasks onto the machines of the HC system. Many different mapping heuristics have been proposed in recent years. However, most of these heuristics suffer from several limitations. One of these limitations is the performance degradation that results from using outdated global information about the status of all machines in the HC system. This paper proposes a new heuristic which addresses this limitation by only requiring partial information in making the mapping decisions. Simulation results show that our heuristic performs very competitively while requiring dramatically less information.*

Keywords: cluster computing, affinity scheduling, linear programming

1 Introduction

Clusters of commodity computers are rapidly gaining acceptance as the preferred way to construct large computing platforms for applications with extensive computer needs [16]. These distributed heterogeneous computing (HC) systems are constructed by networking various machines that have varying capabilities in order to execute a set of tasks. Such systems form the building blocks for grids which are becoming very successful in managing and organizing an institution's computing resources [5].

When a new task arrives to the HC system, it is the responsibility of the RMS to assign the task to a machine and govern the execution of the task [12, 16]. Several mapping heuristics for HC systems have been suggested. This paper proposes a new mapping heuristic that has several advantages over the other heuristics. Throughout this paper, we refer to our new heuristic as the Linear Programming Based Affinity Scheduling (LPAS) heuristic.

It is necessary for a mapping heuristic to guarantee the stability of the HC system, if the system can be stabilized. For instance, the heuristic must prevent any severe imbalance in load across the machines. The LPAS heuristic achieves this property by solving an allocation linear programming problem that provides an allocation of machines to the arriving tasks such that, for stable systems, stability is guaranteed. In addition to guaranteeing stability, the LPAS heuristic also achieves competitive performance in terms of the mean task response time.

In several heuristics, the mapper is assumed to have up-to-date information about the status of all machines (or at least a large number of them). This increases the complexity of the heuristic since the mapper needs to examine a large number of machines. Furthermore, it could be practically difficult to implement these heuristics. Another problem is that the supplied information can be out of date resulting in performance degradation. As observed in [15], this is a major limitation of heuristics which attempt to exploit global information to balance load too aggressively. The LPAS heuristic addresses the problem by only requiring partial information about the status of the HC system at the mapping points.

Our workload model is described in Section 3.1. The model is applicable to HC systems having semi-consistent ETC (expected time to compute) matrices [14]. Although some heuristics have attempted to use partial information, they suffer from poor performance in systems having our workload model. Furthermore, the configuring of such heuristics is not trivial. As will be discussed later, the LPAS heuristic achieves good performance with the added advantage of using partial information.

The remainder of this paper is organized as follows. In Section 2, we discuss work on mapping heuristics for HC systems. In Section 3.2, we describe several existing mapping heuristics. Then, in Section 4, we present the LPAS heuristic. Simulation results and a comparison of the heuristics are provided in Section 5.

2 Related Work

The problem of mapping tasks onto machines in HC systems is an extremely active field (for example, see [4, 7, 8]). In the literature, several authors refer to the mapping of tasks onto machines as scheduling.

Several mapping heuristics are described and compared in [14]. Our new heuristic can be classified as a dynamic, on-line mode mapping heuristic. The model assumptions in [14] and our assumptions for the HC system are identical. However, the authors in [14] assume, in their simulation experiments, that the expected execution times of a task on all machines are known for all the tasks that arrive for execution. This can be unrealistic in typical HC systems. On the other hand, we assume that the tasks are grouped into classes and only the arrival rates of each class's tasks and execution rates of each machine on each class are known. This assumption is made in several models of cluster and grid environments (such as [6, 13]).

The work of [14] is extended in [13] by looking at a real-world workload and also examining the impact of affinity effects. They compare the performance of several scheduling algorithms. One of these algorithms is similar to the MCT (Minimum Completion Time) algorithm [14]. Another algorithm is a variation on the MCT algorithm that attempts to minimize completion time while taking affinity effects into account. Their experimental results show that varying the MCT algorithm to take affinity effects into account has an improved performance over the MCT algorithm.

Several mapping heuristics require that the mapper continuously obtains information from the machines at mapping instances. As discussed in Section 1, performance degradation may result due to the significant complexity of the mapping heuristic or due to the effect of outdated information. A small number of heuristics attempt to avoid this effect by supplying the mapper with information from a small subset of the machines. The KPB (k-percent best) heuristic suggested in [14] is one such heuristic. However, it may introduce instability in highly loaded systems and configuring its parameters may not be trivial. Our new heuristic follows the same idea, but has the advantages of better performance and stability guarantees. Also, it has the advantage of providing an explicit method for finding an allocation of the machines to the tasks.

Several other heuristics for HC systems have been proposed. Some heuristics are designed to address the issues of load balancing [10, 11] and fairness [3]. In [19], a general failure modeling framework is used to study the impact of failures on system performance for a wide range of scheduling policies. Other heuristics that go through a discovery phase are suggested in [13]. Such heuristics differ from the heuristics discussed in this paper which assume a priori knowledge of task execution rates on each machine. Finally, other heuristics have different assumptions than the ones herein. For example, one of the assumptions made is the complete independence of tasks. In [9], heuristics are proposed for HC systems where the tasks depend on the existence of their associated files on the target machine and several files can be shared by several tasks.

Our model for an HC system has been studied in the context of queueing analysis. The MCT heuristic is a variation on the MinDrift rule which is shown to perform well in heavy traffic scenarios [17]. A processor allocation policy which corresponds to the MCT heuristic is introduced in [18].

3 Mapping Heuristics

3.1 Overview

In a general HC system, there is a dedicated mapper for assigning incoming tasks to machines. Let the number of machines in the system be M. It is assumed that the tasks are classified into N classes of tasks. Let I be the set of classes and J be the set of machines. Tasks that belong to the same class i have arrival rate α_i. Furthermore, the execution time of a task on a machine depends on the class of the task and the machine. Let $\mu_{i,j}$ be the execution rate for tasks of class i at machine j, hence $\frac{1}{\mu_{i,j}}$ is the mean execution time for class i tasks at machine j. Let α be the arrival rate vector, the ith element of α is α_i. Also, let μ be the execution rate matrix, having (i, j) entry $\mu_{i,j}$. Several techniques for classifying tasks and obtaining the arrival and execution rates in HC systems exist (see [13]).

The mapping heuristics considered in this paper are on-line mode heuristics [14]. In the on-line mode, a mapping decision is made by the mapper as soon as a task arrives. The tasks are assumed to be independent and atomic. Each new task arriving in the system is assigned to one of the machines immediately upon arrival, and after that, the task can only be executed by the machine to which it is assigned. It is assumed that there is no queueing at the mapper and tasks are queued at the machines to which they are assigned. A First-Come First-Serve (FCFS) scheduling policy is used by the machines.

The on-line mode heuristics assume that the execution rates are known. Furthermore, in most of these heuristics, the mapper uses information supplied by the machines in making mapping decisions. Such information includes, for instance, the machine's expected completion time. Thus, when a task arrives to the HC system, the mapper contacts the machines whose information is needed, and subsequently, the machine supplies the mapper with the requested information.

3.2 Mapping Heuristics

A mapper using the MET (minimum execution time) heuristic assigns an incoming task to the machine that has the least expected execution time for the

task [14]. Thus, when a new task of class i arrives in the system, the mapper assigns it to a machine j such that $j \in \arg\min_{j' \in J} 1/\mu_{i,j'}$. Ties are broken arbitrarily; for instance, a mapper could pick the machine with the smallest index j when more than one machine has the least expected execution time. The MET heuristic does not require the machines to send their expected completion times back to the mapper as tasks arrive, thus the MET heuristic has the advantage of requiring limited communication between the mapper and machines. However, this heuristic can cause severe load imbalance to a degree that the system is unstable. For example, consider an HC system with one arrival stream with rate $\alpha_1 = 6$, and two machines with execution rates $\mu_{1,1} = 5$ and $\mu_{1,2} = 3$, respectively. This system will suffer from load imbalance causing instability if the MET heuristic is used, as no tasks are sent to machine 2. It is easy to see that the system can be stabilized with the given value of α_1.

The MCT (minimum completion time) heuristic assigns an arriving task to the machine that has the earliest expected completion time [14]. Several existing resource management systems use the MCT heuristic or other heuristics that are based on the MCT heuristic, including SmartNet [7, 8].

The MCT heuristic is formally stated as follows. When a task of class i arrives, the mapper assigns it to a machine j such that $j \in \arg\min_{j'} \{1/\mu_{i,j'} + \sum_{i' \in I} Q_{i',j'}/\mu_{i',j'}\}$, where $Q_{i',j'}$ is the number of tasks of class i' that are waiting or executing at machine j', at the time of the arrival of task i. The mapper examines all machines in the HC system to determine the machine with the earliest expected completion time.

The KPB (k-percent best) heuristic attempts to combine elements from both the MET and the MCT heuristics [14]. When a task arrives, the mapper picks the $(kM/100)$ best machines based on the execution times for the task, where $100/M \leq k \leq 100$. Then, the mapper assigns the task to the machine that has the earliest expected completion time in the subset. The KPB heuristic does not only attempt to assign an arriving task to a superior machine based on execution times for the task, it also attempts to avoid assigning the task to a machine that could do better for tasks that arrive later. This foresight is not present in the MCT heuristic. Another advantage of the KPB heuristic is that the mapper needs only to communicate with a subset of the machines, rather than with all machines in the HC system. Note that the KPB heuristic may achieve very poor performance relative to the MCT heuristic in cases where some machines are not among the best $k\%$ for any class.

4 The LPAS Heuristic

Our proposed heuristic is similar to the KPB heuristic in that the mapper needs only to consider a subset of the machines for each class, however, the determination of this subset requires solving a linear programming (LP) problem [1]. Then, the mapper assigns the task to the machine that has the earliest expected completion time in the subset.

The LPAS heuristic requires solving the following allocation LP, where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$ (recall that $\mu_{i,j}$ and α_i are the execution rates and arrival rates for the HC system, respectively). The variables $\delta_{i,j}$ are to be interpreted as the proportional allocation of machine j to class i.

$$\max \ \lambda$$

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

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

$$\delta_{i,j} \geq 0, \ i = 1, \ldots, N, \ j = 1, \ldots, M. \tag{3}$$

The left-hand side of (1) represents the total processing capacity assigned to class i by all machines in the HC system. The right-hand side represents the arrival rate of tasks that belong to class i. Thus, (1) enforces that the total execution allocated for a class should be at least as large as the arrival rate for that class. This constraint is needed to have a stable system. 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 a solution to the allocation LP. The value λ^* can be interpreted as the maximum capacity of the HC system. We define the maximum capacity as follows. Consider an HC system with given values for α_i $(i = 1, \ldots, N)$ and λ^*. If $\lambda^* \leq 1$, then the system is unstable. Thus, the system will be overloaded and tasks queue indefinitely. If, however, $\lambda^* > 1$, then the system can be stabilized if each arrival rate is increased by a factor less than or equal to λ^* (*i.e.* the same HC system with arrival rates $\alpha'_i \leq \lambda^*\alpha_i$, for all $i = 1, \ldots, N$, can be stabilized). The values $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, can be interpreted as the long-run fraction of time that machine j should spend on class i in order to stabilize the system under maximum capacity conditions. Thus, if the arrival rates do not exceed the maximum system capacity, then the system can be stabilized. The $\{\delta^*_{i,j}\}$'s indicate how much each machine should allocate of its capacity to each class. This is guaranteed to stabilize the system [1]. Let δ^* be the machine allocations matrix where the (i, j) entry is $\delta^*_{i,j}$.

The LPAS heuristic can be stated as follows. Given an HC system, solve the allocation LP to find

$\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. When a new task of class i arrives, let S_i denote the set of machines whose $\delta^*_{i,j}$ is not zero ($S_i = \{j : \delta^*_{i,j} \neq 0\}$). The mapper assigns the task to the machine $j \in S_i$ that has the earliest expected completion time among the subset of machines S_i. Again, ties are broken arbitrarily.

The LPAS heuristic considers both the arrival rates and execution rates and their relative values in deciding the allocation of machines to tasks. Furthermore, by solving the allocation LP, the LPAS heuristic provides a systematic approach for setting parameters that guarantee the stability of the system. This is an advantage over the KPB heuristic where figuring the correct value for k may not be a trivial task. The following example clarifies this point and provides some intuition for the LPAS heuristic.

Consider an HC system with two machines and two classes of tasks ($M = 2$, $N = 2$). The arrival and execution rates are given as follows:

$$\alpha = \left[\begin{array}{cc} 2.45 & 2.45 \end{array} \right], \mu = \left[\begin{array}{cc} 9 & 5 \\ 2 & 1 \end{array} \right].$$

We obtain the following solution for the allocation LP:

$$\lambda^* = 1.0204, \delta^* = \left[\begin{array}{cc} 0 & 0.5 \\ 1 & 0.5 \end{array} \right].$$

A mapper using the LPAS heuristic maps all arriving tasks that belong to class 1 to machine 2. At the times of their arrivals, tasks that belong to class 2 are mapped 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, the mapper does not assign any class 1 tasks 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 would be improved significantly if machine 1 only executes class 2 arriving tasks. In fact, the performance of the LPAS heuristic is better than that of the MCT heuristic. For this particular HC system, both the MET heuristic and the KPB heuristic (with $k = 1$) would result in an unstable system. This is because both heuristics map class 2 tasks to machine 1 only, and the system will be unstable since $\alpha_2 > \mu_{2,1}$.

In the LPAS heuristic, the mapper considers a subset of the machines for each class. Ideally, the size of each subset should be much smaller than M. Similar to the KPB heuristic, this has the advantage of requiring less communication between the mapper and the machines. Furthermore, degradation in performance due to outdated information is minimized. To achieve this, the δ^* matrix should contain a large number of elements that are equal to zero. In fact, there could be many solutions to an allocation LP, and a solution with a larger number of zeros 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 [1]):

Proposition 4.1 *There exists a solution to the allocation LP with at least $NM + 1 - N - M$ elements in the δ^* matrix equal to zero.*

The LPAS heuristic can be considered as a dynamic mapping heuristic [4, 14]. As the heuristic only involves solving an LP, it is suited for scenarios when machines are added and/or deleted from the system. On each of these events, one needs to simply solve a new LP and go from there.

5 Simulation Results and Discussion

5.1 Overview

We use simulation to compare the performance of the mapping heuristics. The task arrivals are modeled by independent Poisson random processes, each with rate α_i, $i = 1, \ldots, N$. Unless otherwise stated, the execution times are exponentially distributed with rates $\mu_{i,j}$, where $\frac{1}{\mu_{i,j}}$ represents the mean execution time of a task of class i at machine j, $i = 1, \ldots, N$, $j = 1, \ldots, M$.

Each simulation experiment models a particular HC system, characterized by the values of M, N, α_i, and $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Each experiment simulates the execution of the corresponding HC system for 20,000 time-units.

There are several performance metrics that could be used to compare the performance of the mapping heuristics [14]. We have chosen the long-run average number of tasks in the HC system, L, as a metric for performance comparison. This includes the tasks that are waiting for execution at a particular machine and tasks that are executing.

Each experiment is repeated 30 times. In each experiment, the actual simulated interarrival times and execution times are generated independently. All confidence intervals are at the 95%-confidence level.

5.2 Comparison of the Mapping Heuristics

Table 1 lists simulation results for different HC systems. These systems are discussed in Subsections 5.2.1 to 5.2.5. For each system, the table shows the 95%-confidence interval for L when the corresponding mapping heuristic is used. If a system becomes unstable due to the mapping heuristic used by its mapper, the table just indicates that the system is unstable. Since the MET heuristic results in unstable systems in most of the HC systems in Table 1, we do not include it here. The table also shows the results of using the KPB heuristic with different values for k.

Table 1: Comparison of the Mapping Heuristics

System	MCT	KPB	LPAS
A	$(85.68, 110.23)$	$\underline{k=1}$ Unstable	$(62.56, 82.01)$
B	$(20.05, 21.10)$	$\underline{k=1}$ $(5.65, 5.73)$	$(5.21, 5.26)$
C	$(53.99, 54.98)$	$\underline{k=14}$ $(75.26, 79.13)$	$(47.39, 47.72)$
D	$(22.68, 23.21)$	$\underline{k=2}$ $(14.75, 14.89)$ $\underline{k=3}$ $(11.00, 11.04)$	$(10.55, 10.59)$
E	$(27.71, 28.20)$	$\underline{k=5}$ $(51.65, 55.60)$	$(36.54, 37.07)$
F1	$(19.09, 19.44)$	$\underline{k=4}$ $(20.77, 21.07)$	$(28.71, 29.05)$
F2	$(46.36, 49.49)$	$\underline{k=4}$ $(73.44, 81.75)$	$(34.27, 34.89)$
G	$(37.91, 40.43)$	$\underline{k=4}$ $(42.21, 43.54)$	$(42.05, 43.09)$
H	$(3648.48, 4086.54)$	$\underline{k=5}$ $(888.62, 1319.97)$	$(131.08, 150.15)$
I1	$(64.20, 66.32)$	$\underline{k=14}$ $(86.65, 94.15)$	$(50.83, 38)$
I2	$(41.56, 41.82)$	$\underline{k=14}$ $(53.69, 55.19)$	$(40.57, 40.69)$

5.2.1 Small Systems

System A in Table 1 is the system that is discussed in Section 4. This is a highly loaded system as shown by the large values for L. As shown in the table, the MCT heuristic performs poorly with respect to the LPAS heuristic. This is because the MCT heuristic assigns some class 1 tasks to machine 1, although it is more advantageous to dedicate machine 1 for class 2 tasks.

System B is another HC system, where $M = 2$ and $N = 2$. The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 5 & 8 \end{bmatrix}, \mu = \begin{bmatrix} 8 & 3 \\ 4 & 10 \end{bmatrix}.$$

A solution to the allocation LP is:

$$\delta^* = \begin{bmatrix} 0.8333 & 0 \\ 0.1667 & 1 \end{bmatrix}.$$

As indicated by the δ^* matrix, the LPAS heuristic assigns all class 1 tasks to machine 1. Thus, machine 2 becomes dedicated to execute class 2 tasks. This results in significant performance improvement since, in this case, class 2 tasks have a higher arrival rate and they run much faster on machine 2 than on machine 1.

5.2.2 Large Cluster Systems

System C is a large system with $M = 30$ and $N = 3$. The machines are grouped into four groups, and each group consists of machines with identical performance. Thus, if two machines are in the same group, then they have the same execution rates. Table 2 shows the execution rates of the groups.

Table 2. Execution rates for System C

	Group			
Task	P	Q	R	S
1	8	4	4	4
2	1	4	1	2
3	4	2	8	4

Groups P, Q, R, and S, consist of 10 machines, 9 machines, 6 machines, and 5 machines, respectively. As Table 2 shows, the groups vary in performance. For instance, a machine in group P is twice as fast as a machine in group S on tasks of class 1, however, for tasks of class 2, the opposite is true. The arrival rates are given by $\alpha = [45\ 45\ 40]$.

Since machines that belong to the same group have identical values for δ^*, we represent the δ^* matrix as in Table 3. Note that the δ^* matrix contains 44 elements that are equal to zero.

Table 3. The machine allocations matrix for System C

	Group			
Task	P	Q	R	S
1	0.6270	0	0	0
2	0.3730	0.3730	0.0712	1
3	0	0	0.9288	0

As shown in Table 1, the LPAS heuristic achieves the best results. Note that the KPB heuristic is unstable for $k < 14$.

5.2.3 Task and Machine Heterogeneity

Systems D to G model different kinds of HC system heterogeneity. Machine heterogeneity refers to the average variation along the rows, and similarly task heterogeneity refers to the average variation along the columns [2]. Heterogeneity can be classified into high heterogeneity and low heterogeneity. Based on this, we simulate the following four categories for heterogeneity [2, 14]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo). Due to space limits, we only provide the arrival and execution rates for System E.

System D models a HiHi system with $M = 7$ and $N = 4$. The LPAS heuristic outperforms the other heuristics. It maps the tasks of each class to at most two machines, except for class 2 tasks that are mapped to four machines. The LPAS heuristic has a better performance than the KPB heuristic with $k = 3$.

System E is a LoHi system. The system contains 7 machines and 4 classes. The arrival and execution rates are given by $\alpha = [10\ 10\ 8\ 8]$ and μ given by

$$\begin{bmatrix} 2.2 & 7 & 10.25 & 1 & 5.7 & 0.5 & 12 \\ 1.95 & 7.05 & 9.78 & 0.95 & 5.65 & 0.56 & 11.85 \\ 2 & 7.25 & 10.02 & 0.98 & 5.75 & 0.67 & 11.8 \\ 2.05 & 6.75 & 9.99 & 1.02 & 5.82 & 0.49 & 12.05 \end{bmatrix}.$$

The associated δ^* matrix is:

$$\delta^* = \begin{bmatrix} 1 & 0 & 0.8433 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9151 \\ 0 & 1 & 0.0754 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.0813 & 1 & 1 & 0 & 0.0849 \end{bmatrix}.$$

The MCT heuristic has the best performance for System E. This is not an unexpected result as suggested by the following argument. Due to the very low task heterogeneity of system E, one can think of it as a system with one class of arriving tasks ($\alpha = [36]$) and the execution rate of each machine is the average of the execution rates of the four classes in System E on the machine, $\mu = [2.05\ 7.0125\ 10.01\ 0.9875\ 5.73\ 0.555\ 11.925]$. In this case, assigning an arriving task to the machine that has the minimum expected completion time (the MCT heuristic) is the best strategy. In fact, solving the allocation LP corresponding to the similar system above results in the following value for δ^*: [1 1 1 1 1 1 1]. Thus, in this case, the LPAS heuristic reduces to the MCT heuristic.

Even though the MCT heuristic is the best heuristic for System E, the LPAS heuristic has the advantage of mapping each class to a smaller number of machines. The LPAS heuristic performs much better than the KPB heuristic even for $k = 5$. The KPB heuristic is unstable for $k < 5$.

Systems $F1$ and $F2$ are HiLo systems ($M = 7$, $N = 4$). Both have the same execution rates and only differ in the arrival rates vector α. Due to the very low machine heterogeneity of both systems, one can think of them as consisting of identical machines. The LPAS heuristic compares well with the other heuristics in HiLo environments, and it achieves the best performance in many such systems as in System $F2$.

System G is a LoLo system with $M = 7$ and $N = 4$. The MCT heuristic has slightly better performance than the other heuristics. The KPB heuristic ($k = 4$) has performance close to that of the LPAS heuristic, however, the mapper is required to obtain the expected completion times from four machines at each task arrival as compared to at most 3 machines in the case of the LPAS heuristic.

5.2.4 Special Systems

Consider System H with $M = 7$ and $N = 4$. The arrival and execution rates are given by $\alpha = [8.5\ 8.5\ 9.6\ 8.6]$ and μ given by

$$\begin{bmatrix} 5 & 5.02 & 4.95 & 0.001 & 4.7 & 5.2 & 5.25 \\ 0.001 & 5.09 & 4.9 & 4.92 & 5 & 5.13 & 5.14 \\ 4.45 & 5 & 0.001 & 4.45 & 4.9 & 5 & 5.1 \\ 5.02 & 4.95 & 5 & 5.02 & 5.25 & 4.75 & 0.001 \end{bmatrix}.$$

The system contains a few machines that have very poor performance when executing tasks that belong to particular classes. While such values would most likely not arise in practice, we have chosen these values to magnify the point that assigning a task to a machine that is very poor executing its class may result in significant performance degradation. Since the MCT heuristic does not prevent this from happening, it can result in very poor performance. The LPAS heuristic is the best heuristic for System H, and the δ^* matrix is as follows:

$$\delta^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.6881 & 0 \\ 0 & 0.0824 & 0.3344 & 1 & 0 & 0.3110 & 0 \\ 0 & 0.9176 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0.6656 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The KPB heuristic performs poorly and is only stable for $k \geq 5$. For $k < 5$, instability results. For $k \geq 5$, the system becomes stable, however the performance is poor and it worsens as k increases since it becomes more likely that some tasks are assigned to the poor machines.

5.2.5 Other Execution-time Distributions

To test the effect of execution-time distribution on the performance of the mapping heuristics, all of the previous experiments were re-run with non-exponential execution-time distributions. In particular, two distributions were used to study lower and higher variances than the exponential case: the first is a deterministic distribution with a mean execution time $\frac{1}{\mu_{i,j}}$ for machine j executing class i tasks, and the second is a hyper-exponential distribution with mean $\frac{1}{\mu_{i,j}}$ for the execution times and twice the variance as the exponential case.

Our results indicate that the relative performance of the heuristics has not been affected by the execution-time distribution. System $I1$ has the same configuration as system C, but with a hyper-exponential execution-time distribution. System $I2$ also has the same configuration as system C, but with a deterministic execution-time distribution. Table 1 shows the performance of the different mapping heuristics for Systems $I1$ and $I2$. For the KPB heuristic, both systems are unstable for $k < 14$.

6 Acknowledgements

The first author was supported by an Ontario Graduate Scholarship in Science and Technology. This research is also supported by grants from the Natural Sciences and Engineering Research Council of Canada.

References

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

[2] R. Armstrong. Investigation of effect of different run-time distributions on SmartNet performance. Master's thesis, NAVAL POSTGRADUATE SCHOOL, 1997.

[3] A. Arpaci-Dusseau and D. Culler. Extending proportional-share scheduling to a network of workstations. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1061–1070, 1997.

[4] T. Braun, H. Siegel, N. Beck, L. Bni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 330–335, 1998.

[5] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[6] H. Franke, J. Jann, J. E. Moreira, P. Pattnaik, and M. A. Jette. An evaluation of parallel job scheduling for ASCI Blue-Pacific. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 11–18, 1999.

[7] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Proceedings of the 7th Heterogeneous Computing Workshop*, pages 184–199, 1998.

[8] R. Freund, T. Kidd, and L. Moore. SmartNet: a scheduling framework for heterogeneous computing. In *Proceedings of the 2nd International Symposium on Parallel Architectures, Algorithms and Networks*, pages 514–521, 1996.

[9] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files on heterogeneous master-slave platforms. *Journal of Systems Architecture*, 52(2):88–104, 2006.

[10] K. Huang and H. Chang. Performance evaluation of load sharing policies on computing grid. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 217–223, 2005.

[11] H. D. Karatza and R. C. Hilzer. Parallel and distributed systems: load sharing in heterogeneous distributed systems. In *Proceedings of the 34th Conference on Winter Simulation*, pages 489–496, 2002.

[12] A. Keller, M. Brune, and A. Reinefeld. Resource management for high_performance PC clusters. In *Proceedings of the 7th International High-Performance Computing and Networking Conference*, pages 270–280, 1999.

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

[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*, pages 30–44, 1999.

[15] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel Distributed Systems*, 11(1):6–20, 2000.

[16] T. Sterling, E. Lusk, and W. Gropp, editors. *Beowulf Cluster Computing with Linux*. MIT Press, Cambridge, MA, USA, 2003.

[17] A. Stolyar. Optimal routing in output-queued flexible server systems. *Probability in the Engineering and Information Sciences*, 19(2):141–189, 2005.

[18] K. Wasserman, G. Michailidis, and N. Bambos. Optimal processor allocation to differentiated job flows. *Performance Evaluation*, 63(1):1–14, 2006.

[19] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. Performance implications of failures in large-scale cluster scheduling. In *Proceedings of the Job Scheduling Strategies for Parallel Processing Workshop*, pages 233–252, 2004.