Parallel and Distributed Computing Practices, Special Issue on Algorithms, Systems and Tools for High Performance Computing, accepted, to appear in 2004.

UTILIZATION-BASED TECHNIQUES FOR STATICALLY MAPPING HETEROGENEOUS APPLICATIONS ONTO THE HIPER-D HETEROGENEOUS COMPUTING SYSTEM[*]

SHOUKAT ALI[†], JONG-KOOK KIM[†], YANG YU[‡], SHRIRAM B. GUNDALA[‡], SETHAVIDH GERTPHOL[‡], HOWARD JAY SIEGEL[§¶], ANTHONY A. MACIEJEWSKI[§], AND VIKTOR PRASANNA[‡]

Abstract. This research investigates the problem of allocating a set of heterogeneous applications to a set of heterogeneous machines connected together by a high-speed network. The proposed resource allocation heuristics were implemented on the High Performance Distributed Computing Program's (HiPer-D) Naval Surface Warfare Center testbed. The goal of this study is to design static resource allocation heuristics that balance the utilization of the computation and network resources while ensuring very low failure rates. A failure occurs if no allocation is found that allows the system to meet its resource and quality of service constraints. The broader goal is to determine an initial resource allocation that maximizes the time before run-time re-allocation is required for managing an increased workload. This study proposes two heuristics that perform well with respect to the load-balancing and failure rates. These heuristics are, therefore, very desirable for HiPer-D like systems where low failure rates can be a critical requirement.

1. Introduction. With the widespread use of increasingly powerful commercial off-the-shelf (COTS) products, some system designers have started a shift from custom development to COTS-based systems to achieve lower costs and more flexible systems [22, 26]. However, to use COTS-based systems effectively as parts of a larger system, one needs to exploit the heterogeneity in processor speeds, memory structures, specialized hardware capabilities, etc., that most likely will be present in different COTS products. Heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to meet the requirements of widely varying application mixtures and to maximize the combined performance or cost-effectiveness, e.g., [11, 16, 20, 34].

An important research problem in HC is mapping, i.e., how to assign resources to applications so as to maximize some performance criterion without violating any resource and quality of service constraints. This research addresses the problem of mapping in a High Performance Distributed Computing Program (HiPer-D) [26] like HC system consisting of heterogeneous sets of sensors, applications, machines, and actuators. Each sensor produces data sets periodically at a fixed rate (the rate may be different for different sensors), and these data sets are input into applications. The applications process the data sets and send the output to other applications or to actuators. Each application is required to finish processing a given data set before the next one arrives (this is the throughput constraint on each application). Each sensor can be characterized by the amount of "load" constituted by a given data set (e.g., number of objects in the data set to be processed). The system is expected to operate at a given value of *initial* workload (i.e., the set of sensor load values). However,

[*]This research was supported by the DARPA/ITO Quorum Program through the Office of Naval Research under Grant No. N00014-00-1-0599, and by the Colorado State University George T. Abell Endowment. Some of the equipment used was donated by Intel and Microsoft.

[†]Purdue University, School of Electrical and Computer Engineering, West Lafayette, IN 47907-1285 ({alis, jongkook}@purdue.edu).

[‡]University of Southern California, Department of Electrical Engineering, Los Angeles, CA 90089-2560 ({yangyu, gundala, gertphol, prasanna}@halcyon.usc.edu).

Colorado State University, [§]Department of Electrical and Computer Engineering, [¶]Department of Computer Science, Fort Collins, CO 80523-1373 ({hj, aam}@colostate.edu).

unpredictable changes in the initial sensor loads are likely over time as the content of data sets generated by the sensors changes. This causes unpredictable increases in the execution and communication times for different applications. This means that, at a certain value of the increased workload, some applications may not be able to process a given data set before the next one arrives, causing a throughput violation. Such increases in the workload could require a run-time re-mapping (re-mapping in real time) to manage the increased workload.

The goals of this research are to design a mapping heuristic that (1) produces an initial mapping that maximizes the time before a run-time re-mapping is required, and (2) has a very small "failure rate." It is possible that when a given heuristic is executed for a particular scenario, the heuristic might not be able to find a mapping (e.g., if the heuristic is left with an application that needs more CPU utilization than is available on any machine). This may happen because no feasible mapping exists or because the heuristic made some "bad" assignments early on in its operation so as to leave not enough capacity on the machines for the remaining applications. The failure rate for a given heuristic is defined in this research as the ratio of the number of instances in which the heuristic fails to find a resource allocation to the total number of times the heuristic was executed in different scenarios.

In an ideal situation, one would know the dependence of the computation and communication times of different applications on changes in the workload from its initial state, and use that information to determine a mapping that could meet the first goal. Instead, the information provided was, for each application, the worst-case values of the *minimum* CPU, input network link, and output network link utilizations that the application requires on a given machine to satisfy the throughput constraint at the *initial workload*. This research uses minimizing $U_{\max}$, the utilization of the most utilized computation or network resource, as an approximate approach to reaching the first goal. This approach is reasonable because, generally, as the load from a given sensor increases, the applications that receive data sets from that sensor will utilize more resources. If the system is minimally utilized initially, it can better "absorb" such increases in the workload.

The contribution of this research is the design of two mapping heuristics, MIP* and HRA Max-min, for the initial allocation of resources to applications in the HiPer-D environment. MIP* and HRA Max-min are compared with several other heuristics from the related literature by using simulation experiments, and are seen to outperform the other heuristics, particularly for HC systems with high heterogeneity. The heuristics perform well in the sense that they have very small failure rates, and generate mappings with low values of $U_{\max}$ (as compared to other heuristics). These heuristics are, therefore, very desirable for HiPer-D like systems where increases in workload are likely (assuming that systems that are minimally utilized initially can "absorb" such increases) and low failure rates can be a critical requirement. Note that even though HiPer-D is a real-time system, the contribution of this research is not in guaranteeing the real-time operation of the system, but in designing an *initial* mapping (to be used when the system is first started) that delays the need for real-time, dynamic, on-line re-mapping due to changes in the workload that will cause a throughput violation.

The heuristics discussed in this research are "static" mapping heuristics. Static mapping is performed when the applications are mapped in an off-line planning phase [10], e.g., when a system is first started up and a mapping is needed to ensure that all quality of service (QoS) constraints will be met for a given initial workload. The mapping problem has been shown, in general, to be NP-complete [14, 17, 29]. Thus, the development of heuristic techniques to find near-optimal mappings is an active area of research, e.g., [2, 7, 10, 11, 16, 18, 21, 33, 35, 41].

MSHN (Management System for Heterogeneous Networks) is a collaborative research effort among Colorado State University, Purdue University, the University of Southern California, NOEMIX, and the Naval Postgraduate School [27]. It was supported by the DARPA/ITO Quorum Program. One objective of MSHN is to design and evaluate mapping heuristics for different types of HC environments, including the COTS-based HiPer-D environment at the Naval Surface Warfare Center (NSWC) [26]. A specific example of a HiPer-D subsystem is shown in Figure 1.1. Fire Sim 21, OTH (Over the Horizon) Data Server, and ALT (Air Engagement Control Local Track) Data Server send periodic data to the applications. Tacfire, CFF (Call for Fire) Broker, Land Attack Engagement Server, Deconflict Server, Gun Control, and Display Components are the applications to be mapped. The arrows denote communications, and the labels next to them denote the network protocols used for communications. The labels in parentheses next to the applications denote the types of machines on which those applications can execute. The HiPer-D system consists of a large number of such subsystems.

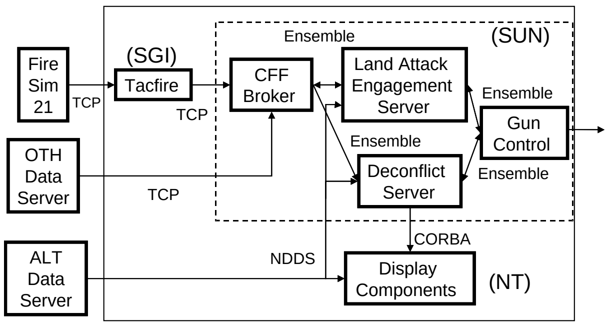


FIG. 1.1. *An example HiPer-D subsystem composed of heterogeneous COTS components.*

The rest of the paper is organized as follows. The system model is described in Section 2. Section 3 presents the static mapping heuristics. The details of the simulation experiments are given in Section 4. Section 5 outlines how this work is related to the previous work in this area. Section 6 concludes the paper.

2. Model. The system consists of heterogeneous sets of sensors, applications, machines, and actuators. Each machine on the network has a full-duplex communication link to a non-blocking switch. The sensors and actuators have unidirectional connections. Each sensor produces data sets periodically, and these data streams are fed into applications. The applications process the data sets and send the output to other applications or to actuators (Figure 1.1).

Let $\underline{\mathcal{M}}$ be the set of machines in the system. Each machine in $\mathcal{M}$ has some "background loads" on its CPU and input/output network links. The background loads on a machine are the utilizations of the CPU, input link, and output link before any applications are mapped on the system.

Let $\underline{\mathcal{A}}$ be the set of applications that need to be mapped. All applications in $\mathcal{A}$ execute continuously to process the periodic inputs that arrive from the sensors or to process input data from predecessor applications. An application can start processing input data as soon as the data is available and the application has finished processing prior data inputs. Recall that each application is required to finish processing a given data set before the next one arrives.

An application is characterized by the worst-case values of the *minimum* CPU, input network link, and output network link utilizations that it requires on a given machine to satisfy the throughput constraint at the *initial workload*. That is, each utilization is a worst-case value for the fraction of the resource required by the given application on a given machine to process a data set (based on its initial load value) within its allowed time period. The utilization of resources information is the required amount of resource to ensure that an application does not violate its throughput constraint. As an example of the relation between the utilization of resource and throughput constraint consider the following. Assume that application A needs 20% of machine 1 for the given initial workload. If application A gets 20% of the CPU cycles on machine 1, then it can meet the throughput constraint. The 20% is the percentage of machine cycles needed (as specified by the sponsor, NSWC) irrespective of the OS multitasking scheduling method within a given machine. This information is the experimental data for the given application, given machine, sensor output rate, and workload.

Let $C(a_i, m_j)$ be the CPU utilization that application a_i requires on machine m_j, with the initial workload, to process a data set within the required time period (based on the rates of its associated sensors). Analogously, let $I(a_i, m_j)$ and $O(a_i, m_j)$ be the input network link and output network link utilizations, respectively, that application a_i requires on machine m_j. Note that the values of $C(a_i, m_j)$, $I(a_i, m_j)$, and $O(a_i, m_j)$ are associated with the *initial workload*.

Let C_j^{bg}, I_j^{bg}, and O_j^{bg} be the background utilizations on the CPU, input network link, and output network link, respectively, of machine m_j. Let $\mathcal{A}_j$ be the set of applications already mapped on machine m_j. Let C_j, I_j, and O_j be the total utilizations on the CPU, input network link, and output network link, respectively, of machine m_j. Then,

$$C_j = C_j^{\text{bg}} + \sum_{a_i \in \mathcal{A}_j} C(a_i, m_j),$$

$$I_j = I_j^{\text{bg}} + \sum_{a_i \in \mathcal{A}_j} I(a_i, m_j), \text{and}$$

$$O_j = O_j^{\text{bg}} + \sum_{a_i \in \mathcal{A}_j} O(a_i, m_j).$$

The total utilization of the most heavily loaded resource (the CPU, input link, or output link) of machine m_j is given by $U_j = \max(C_j, I_j, O_j)$. Ensuring that no resource is more than 100% utilized implies that, for $1 \leq j \leq |\mathcal{M}|$, $U_j \leq 100\%$.

Recall that the performance objective for the mappings in this study is the minimization of $U_{\max}$, the utilization of the most heavily loaded resource, from now on called the maximum utilization. Note that $U_{\max} = \max_{1 \leq j \leq |\mathcal{M}|}(U_j)$.

Including the background load on resources, the utilization of any resource on any machine cannot exceed 100%. This condition ensures that no application violates its throughput constraint. If the resource utilization exceeds 100%, this means that some or all applications on that machine will violate their throughput constraint. If the total utilization of a resource is less than 100%, all applications mapped onto the resource do not violate their throughput constraint.

3. Mapping Heuristics.

This study examines five mapping heuristics, namely: (i) the Min-min heuristic, (ii) the Max-min heuristic, (iii) the host-restriction-aware (HRA) Min-min heuristic, (iv) the host-restriction-aware (HRA) Max-min heuristic, and (v) the Mixed-Integer-Programming-based heuristic (referred to as MIP* in the following text). The HRA Min-min, HRA Max-min, and MIP* heuristics are the three heuristics proposed in this research; Min-min and Max-min are used here for comparison purposes.

The Min-min heuristic is based on [29], and is one of the heuristics implemented in SmartNet [19]. Some variants of the Min-min heuristic were studied in, e.g., [5, 10, 12, 33, 41], and were seen to perform well in many different environments.

Formally, the version of the Min-min heuristic designed for this system can be defined as follows. Let $\underline{U_{r,j}}$ be the total utilization of the most heavily loaded resource (the CPU, input link, or output link) of machine m_j, if the currently unmapped application a_r is mapped on machine m_j. That is,

$$U_{r,j} = \max(C_j + C(a_r, m_j), I_j + I(a_r, m_j), O_j + O(a_r, m_j)).$$

Let $$\underline{U^*} = \min_{i:a_i \in (\mathcal{A} - \bigcup_{j:m_j \in \mathcal{M}} \mathcal{A}_j)} (\min_{j:m_j \in \mathcal{M}} U_{i,j}).$$

The outer minimum in the preceding expression is taken over all unmapped applications. Figure 3.1 shows the pseudo-code used to implement Min-min for this system. Min-min selects the x, y for which $U_{x,y} = U^* \leq 1$, assigns a_x to m_y, adds a_x to $\mathcal{A}_y$, and updates U_y to reflect the assignment. The above process is repeated until all applications are mapped. If $U^* > 1$ in some iteration, a mapping cannot be found with this heuristic. The time complexity of Min-min is $O(|\mathcal{M}||\mathcal{A}|^2)$.

```
(1) do until all applications are mapped
(2)     for each unmapped application $a_r$, find the machine $m_k$ such that
        $U_{r,k} = \min_{j:m_j \in \mathcal{M}} U_{r,j}$ and $U_{r,k} \leq 100\%$
(3)     if no such machine found, this heuristic cannot find a mapping; stop
(4)     from the $(a_r,\ m_k)$ pairs found in step (2), select the pair $(a_x,\ m_y)$ for
        which $U_{x,y} = \min_{(a_r,\ m_k)} U_{r,k}$
(5)     assign the application $a_x$ to the machine $m_y$, mark the application $a_x$ as
        mapped, and update $C_y$, $I_y$, and $O_y$
(6) end do
```

FIG. 3.1. *The Min-min heuristic.*

The Max-min heuristic is similar to the Min-min heuristic, and is also one of the heuristics implemented in SmartNet [19]. It differs from the Min-min heuristic in that now

$$U^* = \max_{i:a_i \in (\mathcal{A} - \bigcup_{j:m_j \in \mathcal{M}} \mathcal{A}_j)} (\min_{j:m_j \in \mathcal{M}} U_{i,j}),$$

and in Figure 3.1, "$\min_{(a_r,\ m_k)}$" in step (4) is replaced with "$\max_{(a_r,\ m_k)}$." Note that the Max-min and Min-min heuristics as described above are directly based on their classical counterparts. This research has adapted the objective function to fit the problem and environment, so that these heuristics could be used here to compare with our proposed heuristics. In some prior studies [10, 33], Min-min had performed better than Max-min in various HC environments. There are cases where Max-min can outperform Min-min. For an example, see [3]. Note that the time complexity of Max-min is the same as that of Min-min, i.e., $O(|\mathcal{M}||\mathcal{A}|^2)$.

The $\underline{\text{host}}$-$\underline{\text{restriction}}$-$\underline{\text{aware}}$ $\underline{\text{Min-min}}$ heuristic ($\underline{\text{HRA Min-min}}$) considers the fact that in many systems a given application may not be able to execute on all machines in the system. This may arise because the application is not compiled for all machines or it requires specialized capabilities available only on select machines. In such systems, the Min-min or Max-min heuristics may fail to find "obvious" mappings for some cases. One such case is shown in Table 3.1, where, for all i, j, $I(a_i, m_j) = O(a_i, m_j) = C_j^{\text{bg}} = I_j^{\text{bg}} = O_j^{\text{bg}} = 0$. The symbol ∞ for an entry $C(a_i, m_j)$ indicates that application a_i cannot execute on machine m_j. Min-min assigns a_1 to m_1 in the first iteration, thereby depriving a_0 of the only machine on which it could execute. Similarly, Max-min first assigns a_0 to m_1, and then assigns a_1 to m_0 in the second iteration, thereby depriving a_2 of the only machine on which it could execute. Hence, both Min-min and Max-min fail to find the obvious mapping (a_0 on m_1, a_1 on m_2, and a_2 on m_0). The HRA Min-min heuristic, described next, does find the obvious mapping.

The HRA Min-min heuristic is shown in Figure 3.2. In each iteration, the heuristic splits the unmapped applications into two sets, and tries to map those sets separately. Let $\underline{U_1^p}$ and $\underline{U_2^p}$ be the two sets of unmapped applications in iteration p. Let $\underline{\mathcal{S}(k)}$ be the set of applications that can map on exactly k machines. In the first iteration, the heuristic splits all applications such that $U_1^1 = \mathcal{S}(1)$ and $U_2^1 = \mathcal{A} - \mathcal{S}(1)$. Then it attempts to map the applications in U_1^1 onto their respective machines. If that partial mapping is not successful, then no mapping exists. If this partial mapping is successful, the heuristic saves the partial mapping, and then tries to map U_2^1 by using Min-min. If the mapping of U_2^1 fails, the heuristic undoes any changes it made to the system while trying to find the mapping of U_2^1, and then moves to the second iteration. In the second iteration, $U_1^2 = \mathcal{S}(2)$ and $U_2^2 = \mathcal{A} - U_1^2 - U_1^1$. In general, before the N-th iteration, $\mathcal{S}(i)$ for $1 \leq i < N$ has been mapped. At that time, $U_1^N = \mathcal{S}(N)$ and $U_2^N = \mathcal{A} - \bigcup_{k=1}^{N} U_1^k$. For all N, HRA Min-min uses Min-min to map U_1^N and U_2^N. (For the case of U_1^1, performing Min-min is equivalent to assigning each application in U_1^1 to the only machine on which it can execute.) Also note that if $\mathcal{S}(N) = \emptyset$ for the N-th iteration, HRA Min-min simply proceeds to the $(N+1)$-th iteration without performing either of the two Min-min operations in iteration N.

A time complexity analysis for HRA Min-min is now presented. The time complexity of determining the set $\mathcal{S}(N)$ for $1 \leq N \leq |\mathcal{M}|$ is $O(|\mathcal{M}||\mathcal{A}|)$. The maximum number of **while** iterations (Line 2, Figure 3.2) is equal to the total number of machines in the system. If a complete mapping is found in the i-th **while** iteration, then the running time for the k-th **while** iteration ($k \leq i$) is given by the sum of the running times of the two Min-min operations performed in iteration k. Mathematically, the running time for the k-th iteration is $O(k|\mathcal{S}(k)|^2) + O(|\mathcal{M}||\mathcal{A} - \bigcup_{j=1}^{k} \mathcal{S}(j)|^2) = O(|\mathcal{M}||\mathcal{A}|^2)$. Given this, HRA Min-min time complexity equals $O(|\mathcal{M}||\mathcal{A}|)+ \sum_{k=1}^{i} \Big(O(|\mathcal{M}||\mathcal{A}|^2)\Big)$. Assume that a complete mapping is found in the first iteration. (This was found fairly common in the experiments performed in this research.) Then, the time complexity for HRA Min-min is equal to that for Min-min. In the worst case, HRA Min-min finds a complete mapping in $|\mathcal{M}|$ **while** iterations, and the time complexity could be up to $O(|\mathcal{M}|^2|\mathcal{A}|^2)$.

The $\underline{\text{host}}$-$\underline{\text{restriction}}$-$\underline{\text{aware}}$ $\underline{\text{Max-min}}$ heuristic ($\underline{\text{HRA Max-min}}$) is similar to the

TABLE 3.1
A scenario showing $C(a_i, m_j)$ values for a system where HRA Min-min finds the obvious mapping, but Min-min and Max-min do not find any feasible mapping.

	m_0	m_1	m_2
a_0	∞	60%	∞
a_1	60%	45%	70%
a_2	50%	∞	∞

```
(1)  $N = 1$
(2)  while ($N \leq |\mathcal{M}|$)
       // if there are any applications that can execute on only $N$ machines, map
       // those applications first
(3)    if ($|\mathcal{S}(N)| > 0$)
(4)      use Min-min to find a mapping for $\mathcal{S}(N)$
(5)      if a mapping for $\mathcal{S}(N)$ is not found, this heuristic has failed; stop
(6)      use Min-min to find a mapping for all of the remaining applications,
         marking each assignment as "speculative"
(7)      if a complete mapping is not found in step (6)
           // roll back - undo all changes to the system data structures,
           // and perform the next iteration
(8)        for each speculative assignment ($a_i$, $m_j$) made in step (6)
(9)          undo the mapping of $a_i$ on $m_j$, and mark application $a_i$ as unmapped
(10)         undo the increases in the CPU, input link, and output link utilizations
             of machine $m_j$ that were caused by speculative mapping of $a_i$ on $m_j$
(11)       $N = N + 1$
(12)     else               // matches the "if" in step (7)
(13)       return mapping
(14)   else               // matches the "if" in step (3)
(15)     $N = N + 1$
(16) end while
```

FIG. 3.2. *The HRA Min-min heuristic.*

HRA Min-min except that it uses the Max-min heuristic instead of Min-min. That is, in steps (4) and (6) in Figure 3.2, Max-min is used instead of Min-min. The time complexity analysis for HRA Max-min is the same as that for HRA Min-min.

The MIP* heuristic is based on the well-researched mixed integer programming (MIP) mathematical technique for optimization [31]. A mathematical programming formulation based on the model in Section 2 is developed to map the applications onto machines. The set $\{x_{ij}\}$, for $1 \leq i \leq |\mathcal{A}|$ and $1 \leq j \leq |\mathcal{M}|$, defines a mapping of applications onto machines such that x_{ij} equals 1 if application a_i is mapped onto machine m_j, but is 0 otherwise. In terms of x_{ij},

$$C_j = C_j^{\text{bg}} + \sum_{1 \leq i \leq |\mathcal{A}|} (x_{ij} \times C(a_i, m_j)),$$

$$I_j = I_j^{\text{bg}} + \sum_{1 \leq i \leq |\mathcal{A}|} (x_{ij} \times I(a_i, m_j)), \text{ and}$$

$$O_j = O_j^{\text{bg}} + \sum_{1 \leq i \leq |\mathcal{A}|} (x_{ij} \times O(a_i, m_j)).$$

Let $\mathcal{M}(a_i)$ be the set of hosts onto which application a_i can be mapped. Figure 3.3 shows the MIP formulation, where U is an auxiliary variable that will equal the minimum value of $U_{\max}$ when the optimization is complete. There are many commercial MIP solvers available that, given enough time, can optimize an MIP problem instance like that in Figure 3.3. The MIP solver, Lindo [38], used in this study employs a branch-and-bound technique to find a solution to a given MIP formulation.

In this study, the objective of the MIP formulation is to minimize $U_{\max}$ based on the constraints that both CPU and network utilizations of each machine are less than or equal to 100%. (However, this approach can be extended to optimize more complex metrics.) The two constraints in the last line in Figure 3.3 force application a_i to be mapped onto exactly one machine in $\mathcal{M}(a_i)$.

Because the above objective function minimizes the maximum utilization (CPU or network) among all machines, the mapping of applications on the less utilized machines may not be necessarily optimized. To try to achieve system-wide optimization, the MIP* heuristic uses an iterative method to solve the problem. In each iteration, MIP* removes from further consideration the most utilized machine and all of the applications mapped thereon, and then attempts to optimize the assignments of the remaining applications on the remaining machines. The mapping is described as a set, $\mathcal{T}$, of $|\mathcal{A}|$ two-tuples, where $\mathcal{T} = \{T_1, \ldots, T_{|\mathcal{A}|}\}$. Each tuple T_i is in the form (a_i, m_j), where $a_i \in \mathcal{A}$ and $m_j \in \mathcal{M}$. Note that there is a tuple (a_i, m_j) in $\mathcal{T}$ if and only if $x_{ij} = 1$. The complete pseudo-code is shown in Figure 3.4. When the stopping condition $\mathcal{M}^* = \emptyset$ is reached, $\mathcal{A}^* = \emptyset$ (unless the heuristic fails to find a mapping). Note, however, that for the performance metric being optimized for this investigation, only one iteration of the repeat-until loop (Figure 3.4, Lines 4-12) is sufficient.

The time complexity of MIP* in the worst case is the same as that for exhaustive search. In real life situations, the running time is usually better than exhaustive search. However, an analysis of the average case time complexity is difficult to derive. See [42] for a detailed complexity analysis of the branch-and-bound methods similar to the one used in the MIP solver used in this study.

given $\mathcal{M}, \mathcal{A}, \{C(a_i, m_j)\}, \{I(a_i, m_j)\}, \{O(a_i, m_j)\}$ and a real number U
find x_{ij} and U
to **minimize** U
subject to $U \leq 100\%$
$\forall m_j \in \mathcal{M},\ C_j \leq U,\ I_j \leq U$, and $O_j \leq U$
$\forall a_i \in \mathcal{A},\ \sum_{j:m_j \in \mathcal{M}(a_i)} x_{ij} = 1$ and $\sum_{j:m_j \notin \mathcal{M}(a_i)} x_{ij} = 0$

FIG. 3.3. *The mixed integer programming formulation.*

In addition to the five heuristics mentioned above, this study also examined a fast greedy heuristic, a random allocation heuristic, and a lower bound (LB) on the maximum utilization. The fast greedy heuristic and the random allocation heuristics are shown in Figure 3.5. Note that, unlike the Min-min or Max-min heuristics, the fast greedy and the random allocation heuristics iterate through the set of applications only once. These two heuristics have time complexities of $O(|\mathcal{M}||\mathcal{A}|)$.

The lower bound on the maximum utilization is calculated by assuming that for all applications $I(a_i, m_j)$ and $O(a_i, m_j)$ are zero, that each application a_i is mapped on the machine m_j where $C(a_i, m_j)$ is minimum over all machines, and that the sum of the utilizations can be divided equally over all of the machines (which, in general,

```
(1)  initialize $\mathcal{T}$ to $\emptyset$
(2)  let $\mathcal{M}^*$ and $\mathcal{A}^*$ denote sets of machines and applications that must be mapped
(3)  initialize $\mathcal{M}^*$ to $\mathcal{M}$ and $\mathcal{A}^*$ to $\mathcal{A}$
(4)  repeat
(5)    using $\mathcal{M}^*$ and $\mathcal{A}^*$, construct a MIP problem instance
         (based on the MIP formulation shown in Figure 3.3)
(6)    solve the MIP problem instance using an MIP solver
(7)    find out the machine $m_x$ that has the highest CPU or network utilization
(8)    for each application $a_i \in \mathcal{A}_x$
         // record the mapping information regarding $m_x$ in $\mathcal{T}$
(9)      add ($a_i$, $m_x$) into $\mathcal{T}$ and delete $a_i$ from $\mathcal{A}^*$
(10)   delete $m_x$ from $\mathcal{M}^*$
(11) until $\mathcal{M}^* = \emptyset$
```

FIG. 3.4. *The MIP* heuristic.*

may not be physically realistic). Specifically,

$$\text{LB} = \Big(\sum_{1 \le i \le |\mathcal{A}|} \min_{1 \le j \le |\mathcal{M}|} C(a_i, m_j) + \sum_{1 \le j \le |\mathcal{M}|} C_j^{\text{bg}} \Big) / |\mathcal{M}|.$$

An example of when this lower bound situation could occur is: (1) all applications that communicate with each other are mapped to the same machine, (2) each application is mapped to its best machine, and (3) the set of applications is such that all machines are equally utilized.

```
    // iterate through the applications in an arbitrary order
(1) for $r = 1$ to $|\mathcal{A}|$
(2)   find the machine $m_k$ such that $U_{r,k} = \min_{j: m_j \in \mathcal{M}} U_{r,j}$ and $U_{r,k} \le 1$
(3)   if no such machine found, this heuristic cannot find a mapping; stop
(4)   assign the application $a_r$ to the machine $m_k$, and update $C_k$, $I_k$, and $O_k$
(5) end for
```

(a)

```
    // iterate through the applications in an arbitrary order
(1) for $r = 1$ to $|\mathcal{A}|$
(2)   find set, $\mathcal{L}$, of machines such that if $a_r$ is mapped on $m_j \in \mathcal{L}$, $U_{r,j} \le 1$
(3)   if $\mathcal{L}$ is empty, this heuristic cannot find a mapping; stop
(4)   assign $a_r$ to a randomly chosen machine $m_k \in \mathcal{L}$, and update $C_k$, $I_k$, and $O_k$
(5) end for
```

(b)

FIG. 3.5. *(a) The fast greedy heuristic. (b) The random allocation heuristic.*

4. Simulation Experiments and Results. In this study, several sets of simulation experiments were conducted to evaluate and compare the heuristics. For all experiments, the number of machines in the system was fixed at ten. Also, it was assumed that every application could execute on at least one machine. That machine was chosen randomly from among all of the machines in the system. For any other machine, the probability that a given application could execute on it was 50%.

The $C(a_i, m_j)$ matrix was generated by sampling a probability distribution $\underline{\mathbf{D}^{\mathbf{C}}}$. The entries in the $C(a_i, m_j)$ matrix were generated to have a mean $\underline{M}^{\mathrm{C}}$, a "task heterogeneity" $\underline{H}^{\mathrm{C}}_{\mathrm{task}}$ (heterogeneity is the standard deviation divided by the mean), and a "machine heterogeneity" $\underline{H}^{\mathrm{C}}_{\mathrm{mach}}$. See [4] for a description of the method used in this study for generating random numbers with these given mean and heterogeneity values. Analogous to the generation of the $C(a_i, m_j)$ values, the $I(a_i, m_j)$ and $O(a_i, m_j)$ values were generated by sampling a probability distribution, $\underline{\mathbf{D}^{\mathbf{IO}}}$, with a mean $\underline{M}^{\mathrm{IO}}$, and task and machine heterogeneities of $\underline{H}^{\mathrm{IO}}_{\mathrm{task}}$ and $\underline{H}^{\mathrm{IO}}_{\mathrm{mach}}$, respectively.

The C_j^{bg} values were generated by sampling a uniform distribution with a mean $\underline{M}^{\mathrm{C}}_{\mathrm{bg}}$ and a heterogeneity $\underline{H}^{\mathrm{C}}_{\mathrm{bg}}$. The I_j^{bg} and O_j^{bg} values were sampled from a uniform distribution with a mean $\underline{M}^{\mathrm{IO}}_{\mathrm{bg}}$ and a heterogeneity $\underline{H}^{\mathrm{IO}}_{\mathrm{bg}}$. For all experiments given here, $H^{\mathrm{C}}_{\mathrm{bg}} = H^{\mathrm{IO}}_{\mathrm{bg}} = 0.1$.

Each experiment consisted of a set of trials. In each trial, new values for $C(a_i, m_j)$, $I(a_i, m_j)$, $O(a_i, m_j)$, C_j^{bg}, I_j^{bg}, and O_j^{bg} were generated by sampling their respective distributions. The number of trials for a given experiment was chosen to give, for the mean of $U_{\max}$, a 95% confidence interval with a "precision" (i.e., the ratio of the half-width of the confidence interval to the mean [30]) of 10% or better.

The results for a selected set of representative experiments are shown in Figures 4.1 to 4.3. (Full results can be seen in [1].) For each heuristic, at most four bars are shown in these figures. The first bar (from the left) shows the average value of $U_{\max}$ found for that heuristic with a 95% confidence interval and a 10% (or better) precision. The second bar shows $\underline{u_{\max}}$, the $U_{\max}$ value averaged only for those trials for which every heuristic successfully found a mapping. The third bar shows the failure rate for the given heuristic. The $\underline{\text{failure}}$ $\underline{\text{rate}}$ ($\underline{\text{FR}}$) of a heuristic is the ratio of the number of trials in which the heuristic could not find a mapping to the total number of trials. Note that the notion of a failure rate does not apply to LB (therefore FR for LB is always shown to be zero in the results given here). When FR for a heuristic is zero, that bar is not shown. The fourth bar shows the average mapping generation time (in milliseconds) of the heuristic averaged only for those trials for which every heuristic successfully found a mapping. Note that the mapping generation time for MIP* was always much larger than that for the other heuristics, and therefore, was not shown in these figures to facilitate graphical comparisons among other heuristics. (The timing data for MIP* is given in the figure captions.)

Consider the significance of the performance metric $u_{\max}$. When FR is zero for all heuristics in a given experiment, then $u_{\max}$ equals the average value of $U_{\max}$, because no trials are excluded for the purpose of calculating $u_{\max}$. For the sake of discussion, assume that, in a certain experiment comparing two heuristics Alg-A and Alg-B, FR is non-zero for Alg-A and is zero for Alg-B. Then, for Alg-B, $u_{\max}$ may differ from the average value of $U_{\max}$ because some trials will be excluded for the purpose of calculating $u_{\max}$. If Alg-B had performed as well in the excluded trials as in the included trials, the difference between $u_{\max}$ and $U_{\max}$ would be zero. However, if Alg-B had performed poorly in the excluded trials, $u_{\max}$ would be smaller than $U_{\max}$. Also, note that by definition, $u_{\max} = U_{\max}$ for Alg-A.

Figure 4.1 shows, for one set of parameters, the relative performances for the

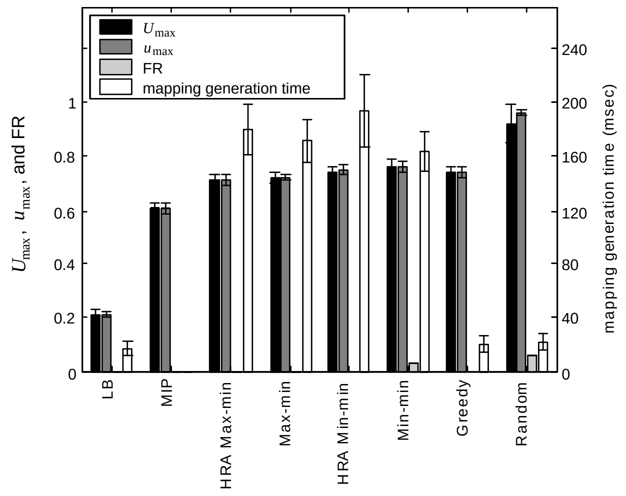


FIG. 4.1. *The relative performance of heuristics in a system with* $|\mathcal{A}| = 40$, $M^{\mathrm{C}} = M^{\mathrm{IO}} =$ *15%,* $H^{\mathrm{C}}_{\mathrm{task}} = H^{\mathrm{C}}_{\mathrm{mach}} = H^{\mathrm{IO}}_{\mathrm{task}} = H^{\mathrm{IO}}_{\mathrm{mach}} =$ *0.3, and* $\mathbf{D^C} = \mathbf{D^{IO}} =$ *gamma. The mapping generation time for MIP* was 30 seconds. A total of 33 trials were performed.*

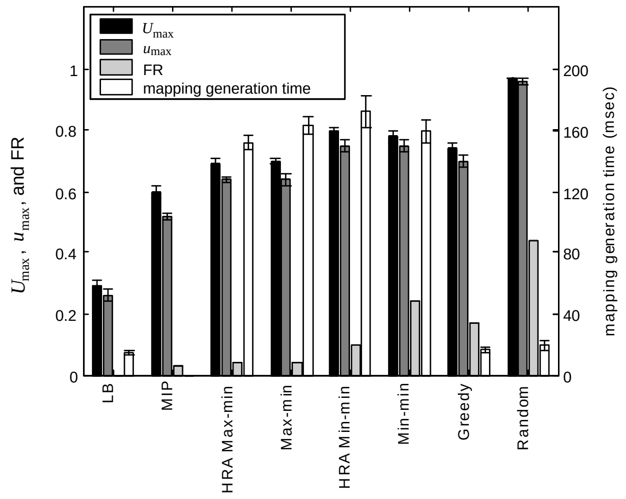


FIG. 4.2. *The effect of increase in the average heterogeneity on the relative performance of heuristics. All parameters are the same as in Figure 4.1, except that* $H^{\mathrm{C}}_{\mathrm{task}} = H^{\mathrm{C}}_{\mathrm{mach}} = H^{\mathrm{IO}}_{\mathrm{task}} = H^{\mathrm{IO}}_{\mathrm{mach}} =$ *0.8. The mapping generation time for MIP* was 60 seconds. A total of 197 trials were performed.*

heuristics discussed in this paper. It can be seen that, for this particular type of HC environment, MIP* outperforms all other heuristics, giving a $U_{\max}$ value 14% smaller than that of the second best heuristic, HRA Max-min. The random allocation heuristic performs the worst. For this particular low heterogeneity HC environment, Max-min, HRA Max-min, HRA Min-min, and Greedy perform comparably with respect to both $U_{\max}$ and FR. However, Figure 4.2 shows that the difference in relative performance increases when the heterogeneity of the application resource utilization increases. Here, the values of $H^{\mathrm{C}}_{\mathrm{task}}$, $H^{\mathrm{C}}_{\mathrm{mach}}$, $H^{\mathrm{IO}}_{\mathrm{task}}$, and $H^{\mathrm{IO}}_{\mathrm{mach}}$ increase to 0.8 from 0.3. The higher heterogeneity values increase the FR for all heuristics. However, this

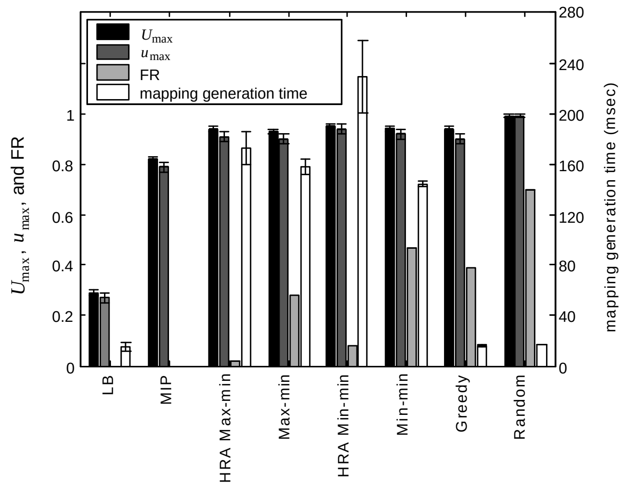


FIG. 4.3. *The effect of increase in the average resource requirement on the relative performance of heuristics. All parameters are the same as in Figure 4.1, except that $M^{\mathrm{C}} = M^{\mathrm{IO}} = 20\%$. The mapping generation time for MIP* was 60 seconds. A total of 90 trials were performed.*

increase is the smallest for MIP*, Max-min, and HRA Max-min. The mapping generation time of HRA Max-min ($\approx$ 150 milliseconds) was smaller than that of MIP* ($\approx$ 30 seconds) by a factor of about 200. Note that the mapping generation time for MIP* was limited to 60 seconds (on an UltraSparc III 750MHz, one gigabyte memory machine running Solaris 5.8). The variation of the quality of mappings generated by MIP* as a function of the mapping generation time will be discussed later.

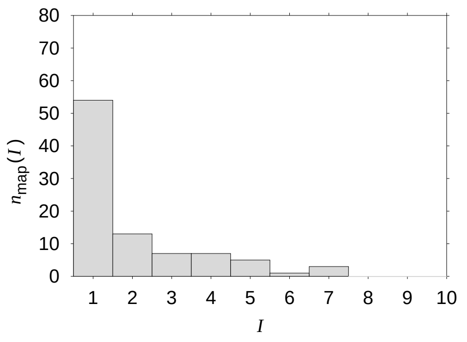
Figure 4.3 shows the change in the relative performance of heuristics when the mean value of the application resource utilization is increased. For this experiment, $M^{\mathrm{C}} = M^{\mathrm{IO}} = 20\%$ (as opposed to $M^{\mathrm{C}} = M^{\mathrm{IO}} = 15\%$ in Figure 4.1). As compared to Figure 4.1, the FR values for all heuristics except MIP* have increased. However, FR for HRA Max-min is very small relatively. MIP* performs the best with respect to both $U_{\max}$ and FR. HRA Max-min follows MIP* very closely with a $U_{\max}$ value 14% larger, and a very small FR value of 2%. Note the very significant difference between the FR of the MIP* and HRA heuristics in comparison to the FR of the other heuristics. The mapping generation time of HRA Max-min ($\approx$ 170 milliseconds) was smaller than that of MIP* ($\approx$ 60 seconds) by a factor of about 350.

Note that unlike Figures 4.1 and 4.2, Figure 4.3 shows statistically significant differences in the mapping generation times for HRA Min-min and Min-min. For this experiment, some relevant data are shown in Table 3 and Figure 4.4. Table 3 gives the distribution of $\mathcal{S}(N)$, the number of applications executable on N machines. It can be seen from Figure 3.2 that Lines 4-13 contribute most towards the mapping generation time of an HRA heuristic. Figure 4.4 shows the distribution of $\overline{n_{\mathrm{map}}(I)}$, the number of trials in which a complete mapping was found in I executions of Lines 4-13 of the HRA heuristic. Note that, depending on the distribution of $\mathcal{S}(N)$, Lines 4-13 might be executed only once or for up to $|\mathcal{M}|$ times during the $|\mathcal{M}|$ iterations of the **while** loop. For example, Lines 4-13 will be executed only once if every application can execute on every machine (because $\mathcal{S}(N) = 0$ for $N \neq |\mathcal{M}|$). The data in Figure 4.4 shows that in approximately 40% of the 90 trials performed, HRA Min-min took more than one iteration. For HRA Max-min this value is only 17%. A higher average number of HRA Min-min iterations explains why HRA Min-min is slower than Min-min in

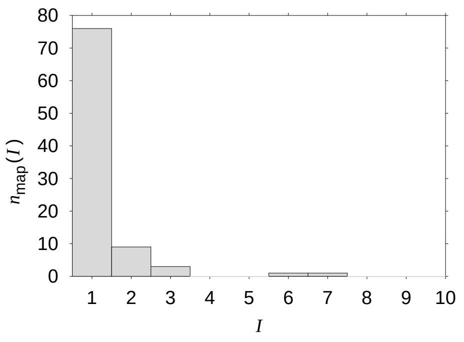
Figure 4.3.

TABLE 4.1
The distribution of $\mathcal{S}(N)$, the number of applications executable on exactly N machines for the experiment in Figure 4.3. The $\mathcal{S}(N)$ values are averaged over 90 trials.

N	1	2	3	4	5	6	7	8	9	10
$\mathcal{S}(N)$	0.4	1.9	4.4	8.2	10.4	8.0	4.8	1.5	0.3	0.0



(a)



(b)

FIG. 4.4. *The distribution of $n_{\text{map}}(I)$, the number of trials in which a complete mapping was found in I executions of Lines 4-13 of the HRA heuristics (Figure 3.2). The data shown corresponds to Figure 4.3, where 90 trials were performed. (a) HRA Min-min. (b) HRA Max-min.*

A discussion of how the length of mapping generation time affects the quality of the mapping generated by MIP* is now presented. As for all iterative search procedures, the quality of the mapping generally improves with the time allocated for the MIP solver (unless the optimal mapping has been found). Experiments were conducted for the maximum mapping generation times of one, two, five, fifteen, 30, and 60 seconds on an UltraSparc III 750MHz, one gigabyte memory machine running Solaris 5.8. Each experiment was defined by the set of parameters in Figure 4.2, and was repeated for 30 trials. The solutions found with a maximum mapping generation time of 30 seconds were very close in quality (as measured by $U_{\max}$) to those found with 60 seconds. For these two times, the average difference in $U_{\max}$ over all trials was less than 3%. For the maximum mapping generation times of one second and 60 seconds, the average difference in $U_{\max}$ over all trials was about 11%. The timing data suggests that for the HC environments discussed here 30 seconds is a reasonable value for the maximum mapping generation time for MIP*.

The experiments conducted so far show that MIP* and HRA Max-min perform the best among all heuristics considered. However, these experiments do not tell how the $U_{\max}$ values given by MIP* and HRA Max-min compare with the optimal $U_{\max}$ value, because finding the optimal mapping for the environment in these experiments is an NP-complete problem. The rest of this section presents some experiments where the simulation was set up so that the optimal value of $U_{\max}$ would be known *a priori*. For these experiments, following special conditions apply. (1) For one randomly chosen application, a_x, $C(a_x, m_j)$ was set to a value of $\underline{K}$ for all m_j. (2) For all other applications, the $C(a_i, m_j)$ values were randomly sampled to ensure that, for all applications that can execute on a given machine m_j, the sum of $C(a_i, m_j)$ values would be at most K. That is,

$$\sum_{(a_i \in \mathcal{A}) \wedge (a_i \neq a_x) \wedge (m_j \in \mathcal{M}(a_i))} C(a_i, m_j) \leq K.$$

The above inequality would hold for all m_j. (3) It was ensured that every application could execute on at least two machines. (4) All communications were ignored, i.e., $I(a_i, m_j)$ and $O(a_i, m_j)$ values were set to zero for all i, j. It is easy to see that such a simulation set-up ensures that the attainable optimal value of $U_{\max} = K$. Figure 4.5 shows that, under the above conditions, MIP* and HRA Max-min perform optimally. Note that the special conditions ensure that FR is zero for any heuristic.

Figure 4.6 shows the results for an experiment where all the special conditions hold, except the second, i.e., $\sum_{(a_i \in \mathcal{A}) \wedge (a_i \neq a_x) \wedge (m_j \in \mathcal{M}(a_i))} C(a_i, m_j)$ could be larger than K. For this experiment, $U_{\max} \geq K$; K is a tight lower bound on $U_{\max}$. It can be seen from Figure 4.6 that MIP* and HRA Max-min have $U_{\max}$ values only slightly larger than the tight lower bound of K (0.81 as opposed to 0.80). Also, the FR values are the smallest for MIP* and HRA Max-min.

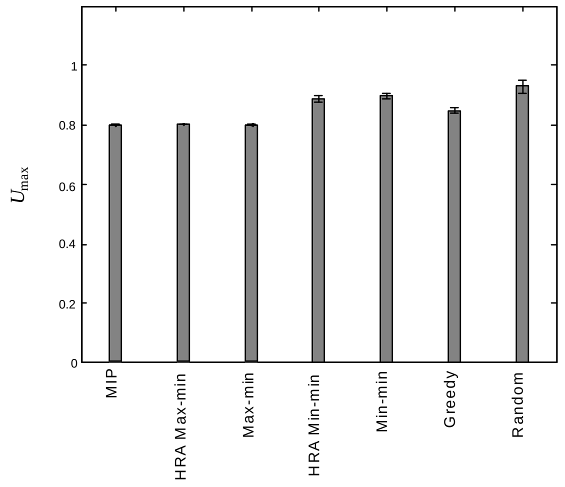


FIG. 4.5. *The relative performance of heuristics for a system where the optimal value of $U_{max} = K = 0.8$. $|\mathcal{A}| = 40$, $M^{\mathrm{C}} = 5\%$, $H^{\mathrm{C}}_{\mathrm{task}} = H^{\mathrm{C}}_{\mathrm{mach}} = 0.3$, and $\mathbf{D^C} =$ gamma. FR was zero for all heuristics, implying that $U_{max} = u_{max}$ for each heuristic. A total of 90 trials were performed.*

5. Related Work. Most of the previous efforts in mapping applications on distributed systems consider the optimization of a given objective function, e.g., overall expected response time or makespan, assuming that the mapping technique *will* find a mapping. As was seen in Section 4, failure rates in high heterogeneity conditions can be a major problem. One way in which our work differs from all of the related efforts given below is that the failure rate analysis for resource allocation in a heterogeneous distributed system is only carried out in our work; other ways are discussed below.

A number of related research efforts in the general topic of mapping and load balancing in heterogeneous distributed computing systems are now discussed. This section is a sampling of related literature and is not meant to be exhaustive.

Several dynamic mapping techniques are discussed in [7, 33], but our work investigates static mapping, and therefore differs from these efforts. The efforts in [2, 21] have similar models to ours, however they differ from our work because they assume

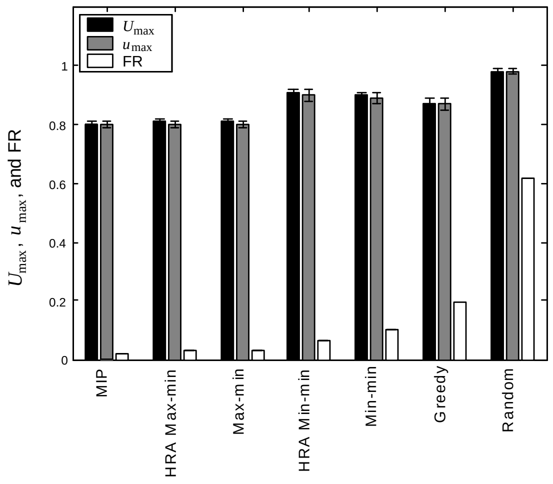


FIG. 4.6. *The relative performance of heuristics for a system where the value of U_{max} is known to be tightly lower-bounded by 0.8. All parameters are the same as in Figure 4.1, except that M^{C} = 30% and $H^{\mathrm{C}}_{\mathrm{task}} = H^{\mathrm{C}}_{\mathrm{mach}}$ = 0.8. A total of 90 trials were performed.*

that the dependence of the computation and communication times of different applications on the workload is known. Our work assumes that only utilization information is available, and, furthermore, that this utilization information is known only for the initial workload. The differences in the available information make the problems in [2, 21] different from ours. The research in [12] considers priorities, deadlines, and multiple versions for a given application, and thus involves a model very different from our model. The work in [10, 41] is perhaps most related to our work. Even though these efforts consider communications differently, have a different performance measure, assume all applications can execute on all machines, and consider single-instance applications rather than continuously executing applications, the problem in [10, 41] has characteristics similar to ours. Therefore, this work uses the Min-min heuristic, which performed among the best in [10, 41], as a basis for our study. We found that there is a significant improvement in performance if Min-min is modified to take into consideration the fact that a given application may not be able to execute on all machines in our study.

Our problem is similar to load balancing problems that have been studied in distributed systems. Many static and dynamic policies have been proposed for solving the load balancing problem (e.g., [6, 8, 9, 15, 24, 25, 32, 37, 39, 40, 43]). Several dynamic policies for load balancing are studied in, e.g., [8, 15, 24, 25]. However, our work belongs to the category of static mapping. Probabilistic static policies with only aggregate information about jobs are studied in, e.g., [6, 32, 40, 43]. However, static utilization information on each individual job (instead of the statistical aggregate information) is assumed to be available in our work.

The effort in [37] studies a distributed computer system that consists of heterogeneous host computers, linked together by a communication network. [37] modifies the system model in [40] by considering, for each machine, a set of applications from a *dedicated traffic type* that must execute on that machine. However, in general, an application in our model can execute on more than one machine, thereby increasing the mapping possibilities and making the problem harder.

The research in [9] studies load balancing in a distributed computer system that consists of heterogeneous host computers, linked together by a communication network. However, it makes the simplifying assumption that the inter-application communication overhead is constant for a given machine, regardless of the particular assignment of the applications. Another simplifying assumption is that all applications present the same "load" to a given processor.

The effort in [39] investigates static mapping techniques to distribute workload in a network of heterogeneous computers. The heterogeneity of the workload itself however is not considered, i.e., it is assumed that a given application has different execution times on different machines, but all applications have the same execution time on a given machine. Our work does not make this assumption, and is more general.

In one way our work complements the efforts in the DeSiDeRaTa [36], TAO [13, 23], and RT ARM adaptive resource manager [28]. These efforts focus on the adaptive management of the system at run time, re-mapping resources in real-time if necessary. Our work tries to configure the system in the lowest possible utilization state (based on the specified initial load) to accommodate the maximum load increase possible before a run-time resource mapping is needed to avoid a throughput constraint violation.

Note that when $U_{\max}$ is minimized, the mapping of applications on the less utilized machines may not be necessarily optimized. Another way our work differs from the related work is that MIP* iteratively removes from further consideration the most utilized machine and all of the applications mapped thereon, and then optimizes the assignments of the remaining applications on the remaining machines. Thus, MIP* employs a procedure for attempting a "system-wide optimization." This is a secondary measure that does not impact the $U_{\max}$ results given in this paper.

6. Conclusions.

This paper examined five static heuristics designed to map applications onto machines in the HiPer-D system. The heuristics were compared under a variety of simulated heterogeneous computing environments. The results from the simulation experiments show that MIP* and HRA Max-min, proposed in this research, perform the best. Both of these heuristics perform well with respect to $U_{\max}$, and also have very small failure rates. A failure occurs if no allocation is found that allows the system to meet its resource and quality of service constraints. These heuristics are, therefore, very desirable for HiPer-D like systems where low failure rates can be a critical requirement. The performance advantage of these heuristics is highest in HC environments with high application and machine heterogeneities, or with high average resource requirements. The results show that, among all heuristics compared in this study, MIP* is the best heuristic for mapping in the HiPer-D environment if the time to generate the mapping is not an issue. However, if the time to generate a mapping should be small, HRA Max-min is the best heuristic.

Acknowledgments: The authors thank the reviewers for their careful comments that were very helpful to us. A preliminary version of portions of this work was presented at the 11th IEEE Heterogeneous Computing Workshop.

REFERENCES

[1] S. Ali, *Robust Resource Allocation in Dynamic Distributed Heterogeneous Computing Systems*, PhD thesis, School of Electrical and Computer Engineering, Purdue University, to appear, 2003.

[2] S. Ali, J.-K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna, *Greedy heuristics for resource allocation in dynamic distributed real-*

time heterogeneous computing systems, in The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002), Vol. II, June 2002, pp. 519–530.

[3] ———, *Utilization-based heuristics for statically mapping real-time applications onto the HiPer-D heterogeneous computing system*, in 11th IEEE Heterogeneous Computing Workshop (HCW 2002) *in the proceedings of 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, Apr. 2002.

[4] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Sedigh-Ali, *Representing task and machine heterogeneities for heterogeneous computing systems*, Tamkang Journal of Science and Engineering, 3 (2000), pp. 195–207, invited.

[5] R. Armstrong, D. Hensgen, and T. Kidd, *The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions*, in 7th IEEE Heterogeneous Computing Workshop (HCW '98), Mar. 1998, pp. 79–87.

[6] S. A. Banawan and N. M. Zeidat, *A comparative study of load sharing in heterogeneous multicomputer systems*, in 25th Annual Symposium on Simulation, Apr. 1992, pp. 22–31.

[7] I. Banicescu and V. Velusamy, *Performance of scheduling scientific applications with adaptive weighted factoring*, in 10th IEEE Heterogeneous Computing Workshop (HCW 2001) *in the proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[8] K. Benmohammed-Mahieddine and P. M. Dew, *A periodic symmetrically-initiated load balancing algorithm for distributed systems*, ACM SIGOPS Operating Systems Review, 28 (1994), pp. 66–79.

[9] N. Bowen, C. Nikolaou, and A. Ghafoor, *On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems*, IEEE Transactions on Computers, 41 (1992), pp. 257–273.

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

[11] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, *Heterogeneous computing: Goals, methods, and open problems*, in The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '01), June 2001, pp. 1–12 (invited keynote paper).

[12] ———, *Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments*, in 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Apr. 2002.

[13] B. S. Doerr, T. Venturella, R. Jha, C. D. Gill, and D. C. Schmidt, *Adaptive scheduling for real-time, embedded information systems*, in 18th IEEE/AIAA Digital Avionics Systems Conference (DASC '99), Vol. 1, Oct. 1999, pp. 2.D.5-1–2.D.5-9.

[14] E. G. Coffman, Jr. (ed.), *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[15] D. L. Eager, E. D. Lazowska, and J. Zahorjan, *Adaptive load sharing in homogeneous distributed systems*, IEEE Transactions on Software Engineering, 12 (1986), pp. 662–675.

[16] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[17] D. Fernandez-Baca, *Allocating modules to processors in a distributed system*, IEEE Transaction on Software Engineering, SE-15 (1989), pp. 1427–1436.

[18] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco, CA, 1999.

[19] R. F. 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 7th IEEE Heterogeneous Computing Workshop (HCW '98), Mar. 1998, pp. 184–199.

[20] R. F. Freund and H. J. Siegel, *Heterogeneous processing*, IEEE Computer, 26 (1993), pp. 13–17.

[21] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel, *A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems*, in 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Apr. 2002.

[22] C. D. Gill, F. Kuhns, D. Levine, D. C. Schmidt, B. S. Doerr, and R. E. Schantz, *Applying adaptive real-time middleware to address grand challenges of COTS-based mission-critical*

real-time systems, in 1st International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems (20th IEEE Real-Time Systems Symposium (RTSS '99)), Nov. 1999.

[23] C. D. Gill, D. L. Levine, and D. C. Schmidt, *The design and performance of a real-time CORBA scheduling service*, Real-Time Systems, 20 (2001), pp. 117–154.

[24] K. K. Goswami, M. Devarakonda, and R. K. Iyer, *Prediction-based dynamic load-sharing heuristics*, IEEE Transactions on Parallel and Distributed Systems, 4 (1993), pp. 638–648.

[25] A. Hac and X. Jin, *Dynamic load balancing in a distributed system using a decentralized algorithm*, in 7th International Conference on Distributed Computing Systems (ICDCS '87), Sep. 1987, pp. 170–177.

[26] R. Harrison, L. Zitzman, and G. Yoritomo, *High performance distributed computing program (HiPer-D)—engineering testbed one (T1) report*, Nov. 1995. Technical Report.

[27] D. A. Hensgen, T. Kidd, D. S. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, *An overview of MSHN: The Management System for Heterogeneous Networks*, in 8th IEEE Heterogeneous Computing Workshop (HCW '99), Apr. 1999, pp. 184–198.

[28] J. Huang, R. Jha, W. Heimerdinger, M. Muhanmmad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhao, and R. Bettati, *RT-ARM: A real-time adaptive resource management system for distributed mission-critical applications*, in IEEE Workshop on Middleware for Distributed Real-Time Systems and Services (18th IEEE Real-Time Systems Symposium (RTSS '97)), Dec. 1997.

[29] O. H. Ibarra and C. E. Kim, *Heuristic algorithms for scheduling independent tasks on nonidentical processors*, Journal of the ACM, 24 (1977), pp. 280–289.

[30] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., New York, NY, 1991.

[31] A. Kaufmann, *Integer and Mixed Programming: Theory and Applications*, Academic Press, New York, NY, 1977.

[32] C. Kim and H. Kameda, *Optimal static load balancing of multi-class jobs in a distributed computer system*, in 10th International Conference on Distributed Computing Systems (ICDCS '90), June 1990, pp. 562–569.

[33] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, *Dynamic mapping of a class of independent tasks onto heterogeneous computing systems*, Journal of Parallel and Distributed Computing, 59 (1999), pp. 107–131.

[34] M. Maheswaran, T. D. Braun, and H. J. Siegel, *Heterogeneous distributed computing*, in Encyclopedia of Electrical and Electronics Engineering, Vol. 8, J. G. Webster, ed., John Wiley, New York, NY, 1999, pp. 679–690.

[35] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, New York, NY, 2000.

[36] B. Ravindran, L. R. Welch, and B. Shirazi, *Resource management middleware for dynamic, dependable real-time systems*, Real-Time Systems, 20 (2001), pp. 183–196.

[37] K. W. Ross and D. D. Yao, *Optimal load balancing and scheduling in a distributed computer system*, Journal of the ACM, 38 (1991), pp. 676–689.

[38] L. Schrage, *LINDO An Optimization Modeling System Text and Software*, The Scientific Press, South San Francisco, 1991.

[39] X. Tang and S. T. Chanson, *Optimizing static job scheduling in a network of heterogeneous computers*, in 29th International Conference on Parallel Processing (ICPP 2000), Aug. 2000, pp. 373–382.

[40] A. N. Tantawi and D. Towsley, *Optimal static load balancing in distributed computer systems*, Journal of the ACM, 32 (1985), pp. 445–465.

[41] M.-Y. Wu, W. Shu, and H. Zhang, *Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems*, in 9th IEEE Heterogeneous Computing Workshop (HCW 2000), May 2000, pp. 375–385.

[42] W. Zhang and R. E. Korf, *Performance of linear-space search algorithms*, Journal of Artificial Intelligence, 79 (1995), pp. 241–292.

[43] Y. Zhang, H. Kameda, and K. Shimizu, *Parametric analysis of optimal static load balancing in distributed computer systems*, Journal of Information Processing, 14 (1991), pp. 433–441.