THE 2004 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP-04), August 2004.

Robust Resource Allocation for Sensor-Actuator Distributed Computing Systems

Shoukat Ali[†], Anthony A. Maciejewski[‡], Howard Jay Siegel[‡§], and Jong-Kook Kim[◇]

[†]University of Missouri-Rolla
Dept. of Electrical and Computer Engineering
Rolla, MO 65409-0040 USA
shoukat@umr.edu

Colorado State University
[‡]Dept. of Electrical and Computer Engineering
[§]Dept. of Computer Science
Fort Collins, CO 80523-1373 USA
{hj, aam}@colostate.edu

[◇]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
jongkook@purdue.edu

Abstract

This research investigates two distinct issues related to a resource allocation: its robustness and the failure rate of the heuristic used to determine the allocation. The target system consists of a number of sensors feeding a set of heterogeneous applications continuously executing on a set of heterogeneous machines connected together by high-speed heterogeneous links. There are a number of quality of service (QoS) constraints that must be satisfied. A heuristic failure occurs if the heuristic cannot find an allocation that allows the system to meet its QoS constraints. The system is expected to operate in an uncertain environment where the workload, i.e., the load presented by the set of sensors, is likely to change unpredictably, possibly invalidating a resource allocation that was based on the initial workload estimate. The focus of this paper is the design of a static heuristic that: (a) determines a robust *resource allocation, i.e., a resource allocation that maximizes the allowable increase in workload until a run-time reallocation of resources is required to avoid a QoS violation, and (b) has a very low* failure rate.

This study proposes a heuristic that performs well with respect to the failure rates and robustness to unpredictable workload increases. This heuristic is, therefore, very desirable for systems where low failure rates can be a critical requirement and where unpredictable circumstances can lead to unknown increases in the system workload.

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.

1. Introduction

This paper investigates the problem of robust resource allocation in a class of heterogeneous computing (HC) systems. An HC system in this class consists of heterogeneous sets of sensors, continuously executing applications, machines, network links, and actuators, and has a number of quality of service (QoS) constraints that must be satisfied during the operation of the system. The system is configured with an initial mapping (i.e., allocation of resources to applications) that is used when the system is first started. The initial mapping attempts to optimize a robustness criterion while ensuring that all QoS constraints will be met for a given initial system workload (i.e., the load associated with the set of initial sensor outputs).

For the particular kind of HC system being considered here, robustness of the initial mapping is an important concern. Generally, these systems operate in an environment that undergoes unexpected changes, e.g., in the system workload, which may cause a QoS violation. Therefore, even though a good initial mapping of applications may ensure that no QoS constraints are violated when the system is first put in operation, dynamic mapping approaches may be needed to reallocate resources during execution to avoid QoS violations.

The general goal of this paper is to delay the *first remapping* of resources required at run time to prevent QoS violations due to variations in the amount of workload generated by the changing sensor outputs. This paper uses a generalized performance metric that is suitable for evalu-

ating an initial mapping for such "robustness" against increases in the workload (a formal definition of robustness and a general procedure to derive it are given in [3]). The initial mapping problem is defined as finding a static mapping (i.e., one found in an off-line planning phase) of a set of applications onto a suite of machines to maximize the robustness against workload, where robustness is defined as the maximum allowable increase in system workload until run-time re-mapping of the applications is required to avoid a QoS violation. The contributions of this research include quantifying a metric for robustness, designing and developing heuristics for mapping the applications so as to optimize the robustness, and evaluating the relative performance of these heuristics for the intended dynamic distributed HC system. The mapping problem has been shown, in general, to be NP-complete [12, 13, 16]. Thus, the development of heuristic techniques to find near-optimal mappings is an active area of research, e.g., [5, 6, 9, 14, 19, 20, 21].

The remainder of this paper is organized in the following manner. Section 2 develops models for the applications and the hardware platform. Section 3 presents a quantitative measure of the robustness of a given mapping of applications to machines. Three heuristics to solve the initial mapping problem are described in Section 4. The simulation experiments and the evaluation of the heuristics are discussed in Section 5. A sampling of some related work is presented in Section 6. Section 7 concludes the paper.

2. System Model

The system consists of heterogeneous sets of sensors, applications, machines, and actuators. Each machine is capable of multi-tasking, executing the applications allocated to it in a round robin fashion. Similarly, a given network link is multi-tasked among all data transfers using that link. Each sensor produces data periodically at a certain rate, and the resulting data streams are input into applications. The applications process the data and send the output to other applications or to actuators. The applications and the data transfers between them are modeled with a directed acyclic graph, shown in Figure 1.

The figure also shows a number of *paths* (enclosed by dashed lines) formed by the applications. A path is a chain of producer-consumer pairs that starts at a sensor (the driving sensor) and ends at an actuator (if it is a trigger path) or at a multiple-input application (if it is an update path). In the context of Figure 1, path 1 is a trigger path, and path 2 is an update path. In a real system, application d could be a missile firing program that produces an order to fire. It needs target coordinates from application b in path 1, and an updated map of the terrain from application c in path 2. Naturally, application d must respond to any output from b, but must not issue fire orders if it receives an output from c alone; such an output is used only to update an internal database. So while d is a multiple input application, the rate at which it produces data is equal to the rate at which the "trigger" application b produces data. That rate, in turn, equals the rate at which the driving sensor, S_1, produces data. The problem specification indicates the path to which each application belongs, and the corresponding driving sensor.

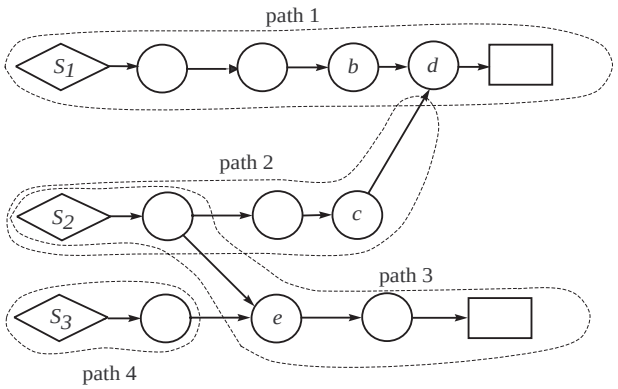

Figure 1. The DAG model for the applications (circles) and data transfers (arrows). The diamonds and rectangles denote sensors and actuators, respectively. The dashed lines enclose each path formed by the applications.

Let $\mathcal{P}$ be the set of all paths, and $\mathcal{P}_k$ be the list of applications that belong to the k-th path. Note that an application may be present in multiple paths. Let $\mathcal{A}$ be the set of applications.

The sensors constitute the interface of the system to the external world. Let the maximum periodic data output rate from a given sensor be called its output data rate. The minimum throughput constraint states that the computation or communication time of any application in $\mathcal{P}_k$ is required to be no larger than the reciprocal of the output data rate of the driving sensor for $\mathcal{P}_k$. For application $a_i \in \mathcal{P}_k$, let $R(a_i)$ be set to the output data rate of the driving sensor for $\mathcal{P}_k$. In addition, let T^{c}_{ij} be the computation time for application a_i allocated to machine m_j. The "c" in the superscript denotes "computation." Also, let T^{t}_{ip} be the time to send data from application a_i to application a_p. The "t" in the superscript denotes "transfer." Because both machines and communications are assumed to be multi-tasked, T^{c}_{ij} and T^{t}_{ip} will depend on the level of multi-tasking (i.e., the number of applications assigned to a machine or the number of communications assigned to a link). See [1] for further details of the computation and communication models used here.

The maximum end-to-end latency constraint states that, for a given path $\mathcal{P}_k$, the time taken between the instant the driving sensor outputs a data set until the instant the actuator or the multiple-input application fed by the path receives the result of the computation on that data set must be no greater than a given value, $L_k^{\max}$. Let L_k be the actual (as opposed to the maximum allowed) value of the end-to-end latency for $\mathcal{P}_k$. The quantity L_k can be found by adding the computation and communication times for all applications in $\mathcal{P}_k$ (including any sensor output or actuator input communications). Let $\mathcal{D}(a_i)$ be the set of successor applications of a_i. Then,

$$L_k = \sum_{\substack{i:\ a_i \in \mathcal{P}_k \\ p:\ (a_p \in \mathcal{P}_k) \wedge (a_p \in \mathcal{D}(a_i))}} \left[T_{ij}^{\mathrm{c}} \ + \ T_{ip}^{\mathrm{t}} \right]. \qquad (1)$$

Let λ_z be the output from the z-th sensor in the set of sensors, and be defined as the number of objects present in the most recent data set from that sensor. This system is expected to operate under uncertain outputs from the sensors, requiring that the resource allocation be robust against unpredictable increases in the sensor outputs. The system workload, $\boldsymbol{\lambda}$, is the vector composed of the load values from all sensors. Let $\boldsymbol{\lambda}^{\text{init}}$ be the initial value of $\boldsymbol{\lambda}$, and λ_i^{init} be the initial value of the i-th member of $\boldsymbol{\lambda}^{\text{init}}$.

The computation times of different applications (and the communication times of different data transfers) are likely to be of different complexities with respect to $\boldsymbol{\lambda}$. Assume that the dependence of T_{ij}^{c} and T_{ip}^{t} on $\boldsymbol{\lambda}$ is known (or can be estimated) for all i, p. Then, T_{ij}^{c} and T_{ip}^{t} can be re-expressed as functions of $\boldsymbol{\lambda}$ as $T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda})$, respectively. In general, $T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda})$ will be functions of the loads from all those sensors that can be traced back from a_i. For example, the computation time for application d in Figure 1 is a function of the loads from sensors S_1 and S_2, but that for application e is a function of the S_2 and S_3 loads (but each application has just one driving sensor: S_1 for d and S_2 for e). Then Equation 1 can be used to express L_k as a function of $\boldsymbol{\lambda}$.

3. Performance Goal

This section quantifies the robustness of a mapping [3]. To simplify the presentation, but without loss of generality, it is assumed that $\boldsymbol{\lambda}$ is a continuous variable, and that computation and communication times are continuous functions of $\boldsymbol{\lambda}$. The change in $\boldsymbol{\lambda}$ can occur in different "directions" depending on the relative changes in the individual components of $\boldsymbol{\lambda}$. For example, $\boldsymbol{\lambda}$ might change so that all components of $\boldsymbol{\lambda}$ increase in proportion to their initial values. In another case, only one component of $\boldsymbol{\lambda}$ may increase while all other components remain fixed. Figure 2 illustrates some possible directions of increase in $\boldsymbol{\lambda}$. In Figure 2, $\boldsymbol{\lambda}^{\text{init}} \in \mathbf{R}^2$ is the initial value of the system load. The region enclosed by the axes and the curve C gives the feasible values of $\boldsymbol{\lambda}$, i.e., all those values for which the system does not violate a given QoS constraint. The element of C marked as $\boldsymbol{\lambda}^*$ has the feature that the Euclidean distance from $\boldsymbol{\lambda}^{\text{init}}$ to $\boldsymbol{\lambda}^*$, $||\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^{\text{init}}||$, is the smallest over all such distances from $\boldsymbol{\lambda}^{\text{init}}$ to a point on C. (The symbol $||\cdot||$ stands for the Euclidean norm.) An important interpretation of $\boldsymbol{\lambda}^*$ is that the value $||\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^{\text{init}}||$ gives the largest Euclidean distance that the variable $\boldsymbol{\lambda}$ can move in *any* direction from an initial value of $\boldsymbol{\lambda}^{\text{init}}$ without incurring a QoS violation. *This paper defines $\Delta\boldsymbol{\Lambda} = ||\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^{init}||$ to be the robustness of a mapping, against the system workload, with respect to satisfying the QoS constraints.*

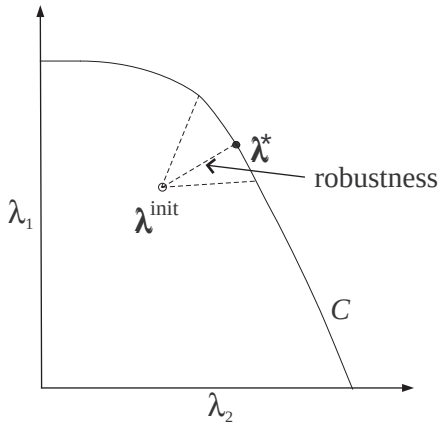

Figure 2. Some possible directions of increase of the system load λ, and the degree of robustness.

A conceptual way of determining $\Delta\boldsymbol{\Lambda}$ is now given. Let $\mathcal{L}_i^{\mathrm{T}}$ be the set of all those $\boldsymbol{\lambda}$ values at which application a_i equals its throughput constraint, i.e., $\mathcal{L}_i^{\mathrm{T}} = \left\{\boldsymbol{\lambda} : T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda}) = 1/R(a_i)\right\} \bigcup \left\{\boldsymbol{\lambda} :\ \forall a_p \in \mathcal{D}(a_i),\ T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda}) = 1/R(a_i)\right\}$. The "T" in the superscript denotes "throughput." Let $\mathcal{L}^{\mathrm{T}}$ be the set of $\boldsymbol{\lambda}$ values at which *any* application equals its throughput constraint, i.e., $\mathcal{L}^{\mathrm{T}} = \bigcup_{a_i \in \mathcal{A}} (\mathcal{L}_i^{\mathrm{T}})$.

Similarly, let $\mathcal{L}_k^{\mathrm{L}}$ be the set of those $\boldsymbol{\lambda}$ values at which path $\mathcal{P}_k$ equals its latency constraint, i.e.,

$$\mathcal{L}_k^{\mathrm{L}} = \{\boldsymbol{\lambda} : \sum_{\substack{i:\ a_i \in \mathcal{P}_k \\ p:\ (a_p \in \mathcal{P}_k) \wedge (a_p \in \mathcal{D}(a_i))}} \left[T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda}) \ + T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda}) \right] = L_k^{\max}\}.$$

The "L" in the superscript denotes "latency." Let $\mathcal{L}^{\mathrm{L}}$ be the set of $\boldsymbol{\lambda}$ values at which *any* path equals its latency constraint, i.e., $\mathcal{L}^{\mathrm{L}} = \bigcup_{\mathcal{P}_k \in \mathcal{P}} (\mathcal{L}_k^{\mathrm{L}})$.

Finally, let $\mathcal{L}$ be the set of $\boldsymbol{\lambda}$ given by $\mathcal{L}^{\mathrm{T}} \bigcup \mathcal{L}^{\mathrm{L}}$. One can then determine $\Delta\boldsymbol{\Lambda}$ by determining the smallest value of $||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||$ over all $\boldsymbol{\lambda} \in \mathcal{L}$. That is,

$$\Delta\boldsymbol{\Lambda} = \min_{\boldsymbol{\lambda} \in \mathcal{L}} ||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||. \quad (2)$$

This research assumes that the optimization problem given in Equation 2 can be solved to find the global minimum. An optimization problem of the form $x^* = \text{argmin}_x f(x)$, subject to the constraint $g(x) = 0$, where $f(x)$ and $g(x)$ are convex and linear functions, respectively, can be solved easily to give the global minimum [8]. Because all norms are convex functions, the optimization problem posed in Equation 2 reduces to a convex optimization problem if $T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda})$ are linear functions. If $T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\mathrm{t}}(\boldsymbol{\lambda})$ functions are not linear, then it is assumed that heuristic techniques could be used to find near-optimal solutions.

4. Heuristic Descriptions

This section develops three greedy heuristics for the problem of finding an initial static allocation of applications onto machines to maximize $\Delta\boldsymbol{\Lambda}$. Greedy techniques perform well in many situations, and have been well-studied (e.g., [16]). One of the heuristics, Most Critical Task First (MCTF), is designed to work well in heterogeneous systems where the throughput constraints are more stringent than the latency constraints. The other heuristic, the Most Critical Path First (MCPF) heuristic, is designed to work well in heterogeneous systems where the latency constraints are more stringent than the throughput constraints.

It is important to note that these heuristics use the $\Delta\boldsymbol{\Lambda}$ value to guide the heuristic search; however, the procedure given in Section 3 for calculating $\Delta\boldsymbol{\Lambda}$ assumes that a complete mapping of all applications is known. During the course of the execution of the heuristics, not all applications are mapped. In these cases, for calculating $\Delta\boldsymbol{\Lambda}$, the heuristics assume that each such application a_i is mapped to the machine where its computation time is smallest over all machines, and that a_i is using 100% of that machine. Similarly for communications where one or two of the applications is unmapped, it is assumed that the data transfer occurs over the highest speed communication link, and that the link is 100% utilized by the data transfer. With these assumptions, $\Delta\boldsymbol{\Lambda}$ is calculated and used in any step of a given heuristic.

Before discussing the heuristics, some additional terms are now defined. Let $\underline{\Delta\boldsymbol{\Lambda}^{\mathrm{T}}}$ be the robustness of the resource allocation when only throughput constraints are considered, i.e, all latency constraints are ignored. Then, $\Delta\boldsymbol{\Lambda}^{\mathrm{T}} = \min_{\boldsymbol{\lambda} \in \mathcal{L}^{\mathrm{T}}} ||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||$. Similarly, let $\underline{\Delta\boldsymbol{\Lambda}^{\mathrm{L}}}$ be the robustness of the resource allocation when only latency constraints are considered. Then, $\Delta\boldsymbol{\Lambda}^{\mathrm{L}} = \min_{\boldsymbol{\lambda} \in \mathcal{L}^{\mathrm{L}}} ||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||$. In addition, let $\underline{\Delta\boldsymbol{\Lambda}_{ij}^{\mathrm{T}}}$ be the robustness of the assignment of a_i with respect to the throughput constraint, i.e., it is the largest increase in load in any direction from the initial value that does not cause a throughput violation for application a_i, either for the computation of a_i on machine m_j or for the communications from a_i to any of its successor applications. Then, $\Delta\boldsymbol{\Lambda}_{ij}^{\mathrm{T}} = \min_{\boldsymbol{\lambda} \in \mathcal{L}_i^{\mathrm{T}}} ||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||$. Similarly, let $\underline{\Delta\boldsymbol{\Lambda}_k^{\mathrm{L}}}$ be the robustness of the assignment of applications in $\mathcal{P}_k$ with respect to the latency constraint, i.e., it is the largest increase in load in any direction from the initial value that does not cause a latency violation for the path $\mathcal{P}_k$. It is given by $\min_{\boldsymbol{\lambda} \in \mathcal{L}_k^{\mathrm{L}}} ||\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{init}}||$.

Most Critical Task First Heuristic: The MCTF heuristic makes one application to machine assignment in each iteration. Each iteration can be split into two phases. Let $\underline{\mathcal{M}}$ be the set of machines in the system. Let $\underline{\Delta\boldsymbol{\Lambda}^*(a_i, m_j)}$ be the value of $\Delta\boldsymbol{\Lambda}$ *if* application a_i is mapped on m_j. Similarly, let $\underline{\Delta\boldsymbol{\Lambda}^{\mathrm{T}*}(a_i, m_j)}$ be the value of $\Delta\boldsymbol{\Lambda}_{ij}^{\mathrm{T}}$ *if* application a_i is mapped on m_j. In the first phase, each unmapped application a_i is paired with its "best" machine m_j such that

$$m_j = \underset{m_k \in \mathcal{M}}{\text{argmax}}(\Delta\boldsymbol{\Lambda}^*(a_i, m_k)). \quad (3)$$

(Note that $\text{argmax}_x f(x)$ returns the value of x that maximizes the function $f(x)$. If there are multiple values of x that maximize $f(x)$, then $\text{argmax}_x f(x)$ returns the set of all those values.) If the RHS in Equation 3 returns a set of machines, $\underline{G(a_i)}$, instead of a unique machine, then $m_j = \text{argmax}_{m_k \in G(a_i)}(\Delta\boldsymbol{\Lambda}^{\mathrm{T}*}(a_i, m_k))$, i.e., the individual throughput constraints are used to break ties in the overall system-wide measure. If $\Delta\boldsymbol{\Lambda}^*(a_i, m_j) < 0$, this heuristic cannot find a mapping. The first phase does not make an application to machine assignment; it only establishes application-machine pairs (a_i, m_j) for all unmapped applications a_i.

The second phase makes an application to machine assignment by selecting one of the (a_i, m_j) pairs produced by the first phase. This selection is made by determining the most "critical" application (the criterion for this is explained later). The method used to determine this assignment in the first iteration is totally different from that used in the subsequent iterations.

Consider the motivation for the special first iteration. Let $\underline{\Delta\boldsymbol{\Lambda}_g}$ be the value of $\Delta\boldsymbol{\Lambda}$ at the end of the g-th iteration. Before the first iteration of the heuristic, all applications are unmapped, and the system resources are entirely unused. With the system in this state, the heuristic selects the pair (a_x, m_y) such that

$$(a_x, m_y) = \underset{\substack{(a_i, m_j) \text{ pairs from} \\ \text{the first phase}}}{\text{argmin}} (\Delta\boldsymbol{\Lambda}^*(a_i, m_j)).$$

The application a_x is then assigned to the machine m_y. It is likely that if the assignment of this application is postponed,

it might have to be assigned to a machine where its maximum allowable increase in the system load is even smaller. (The discussion above does not imply that an optimal mapping must contain the assignment of a_x on m_y.) Experiments conducted in this study have shown that the special first iteration significantly improves the performance.

The criterion used to make the second phase application to machine assignment for iterations 2 to $|\mathcal{A}|$ is different from that used in iteration 1, and is now explained. The intuitive goal is to determine the (a_i, m_j) pair, which if not selected, may cause the most future "damage," i.e., decrease in $\Delta\mathbf{\Lambda}$. Let $\mathcal{M}^{a_i}$ be the ordered list, $\langle m_1^{a_i}, m_2^{a_i}, \cdots, m_{|\mathcal{M}|}^{a_i}\rangle$, of machines such that $\Delta\mathbf{\Lambda}^*(a_i,\ m_x^{a_i}) \geq \Delta\mathbf{\Lambda}^*(a_i,\ m_y^{a_i})$ if $x < y$. Note that $m_1^{a_i}$ is the same as a_i's "best" machine. Let v be an integer such that $2 \leq v \leq |\mathcal{M}|$, and let $r(a_i, v)$ be the percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ if a_i is mapped on $m_v^{a_i}$ (its v-th best machine) instead of $m_1^{a_i}$, i.e.,

$$r(a_i, v) = \frac{\Delta\mathbf{\Lambda}^*(a_i,\ m_1^{a_i}) - \Delta\mathbf{\Lambda}^*(a_i,\ m_v^{a_i})}{\Delta\mathbf{\Lambda}^*(a_i,\ m_1^{a_i})}.$$

Additionally, let $T(a_i, 2)$ be defined such that,

$$T(a_i, 2) = \frac{\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_1^{a_i}) - \Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_2^{a_i})}{\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_1^{a_i})}.$$

Then, in all iterations other than the first iteration, MCTF maps the most critical application, where the most critical application is found using the pseudo-code in Figure 3. The technique shown in Figure 3 builds on the idea of the Sufferage heuristic given in [19].

Two-Phase Greedy Heuristic: This research also proposes a modified version of the Min-min heuristic. Variants of the Min-min heuristic (first presented in [16]) have been studied, e.g., [1, 9, 19, 21], and have been seen to perform well in the environments for which they were proposed. Two-Phase Greedy (TPG), a Min-min style heuristic for the environment discussed in this research, is shown in Figure 4.

Most Critical Path First Heuristic: The MCPF heuristic explicitly considers the latency constraints of the paths in the system. It begins by ranking the paths in the order of the most "critical" path first (defined below). Then it uses a modified form of the MCTF heuristic to map applications on a path-by-path basis, iterating through the paths in a ranked order. The modified form of MCTF differs from MCTF in that the first iteration has been changed to be the same as the subsequent iterations.

The ranking procedure used by MCPF is now explained in detail. Let $\hat{\mathbf{\Lambda}}^{\mathrm{L}}(\mathcal{P}_k)$ be the value of $\Delta\mathbf{\Lambda}_k^{\mathrm{L}}$ assuming that each application a_i in $\mathcal{P}_k$ is mapped to the machine m_j where it has the smallest computation time, and that a_i can use 100% of m_j. Similarly for the communications between the consecutive applications in $\mathcal{P}_k$, where one or two of the applications is unmapped, it is assumed that the data transfer between the applications occurs over the highest speed communication link, and that the link is 100% utilized by the data transfer. The heuristic ranks the paths in

(1) **initialize**: $v = 2$; $\mathcal{F}$ = the set of (a_i, m_j) pairs from the first phase
(2) **for** $v = 2$ to $|\mathcal{M}|$
(3) **if** $\mathrm{argmax}_{(a_i,m_j)\in\mathcal{F}}(r(a_i, v))$ is a unique pair (a_x, m_y)
(4) **return** (a_x, m_y)
(5) **else**
(6) $\mathcal{F}$ = the set of pairs returned by $\mathrm{argmax}_{(a_i,m_j)\in\mathcal{F}}(r(a_i, v))$
(7) **end for**
/* program control reaches here only if no */
/* application, machine pair has been */
/* selected in Lines 1 to 7 above. */
/* $\mathcal{F}$ is now the set of (a_i, m_j) pairs from */
/* the last execution of Line 6*/
(8) **if** $\mathrm{argmax}_{(a_i,m_j)\in\mathcal{F}}(T(a_i, 2))$ is a unique pair (a_x, m_y)
(9) **return** (a_x, m_y)
(10) **else**
(11) arbitrarily select and return an application, machine pair from the set of pairs given by $\mathrm{argmax}_{(a_i,m_j)\in\mathcal{F}}(T(a_i, 2))$

Figure 3. Selecting the most critical application to map next given the set of (a_i, m_j) pairs from the first phase of MCTF.

(1) **do** until all applications are mapped
(2) **for** each unmapped application a_i, find the machine m_j such that $m_j = \mathrm{argmax}_{m_k\in\mathcal{M}}(\Delta\mathbf{\Lambda}^*(a_i, m_k))$; resolve ties arbitrarily
(3) **if** $\Delta\mathbf{\Lambda}^*(a_i, m_j) < 0$, this heuristic cannot find a mapping
(4) from the (a_i, m_j) pairs found above, select the pair(s) (a_x, m_y) such that $(a_x, m_y) = \mathrm{argmax}_{(a_i,m_j)\text{ pairs}}(\Delta\mathbf{\Lambda}^*(a_i, m_j))$; resolve ties arbitrarily
(5) map a_x on m_y
(6) **enddo**

Figure 4. The TPG heuristic.

an ordered list $\langle \mathcal{P}_1^{\text{crit}}, \mathcal{P}_2^{\text{crit}}, \cdots, \mathcal{P}_{|\mathcal{P}|}^{\text{crit}} \rangle$ such that $\hat{\mathbf{\Lambda}}^{\text{L}}(\mathcal{P}_x^{\text{crit}}) \leq \hat{\mathbf{\Lambda}}^{\text{L}}(\mathcal{P}_y^{\text{crit}})$ if $x < y$.

For an arbitrary HC system, one is not expected to know if the system is more stringent with respect to latency constraints or throughput constraints. In that case, this research proposes running both MCTF and MCPF, and taking the better of the two mappings. The Duplex heuristic executes both MCTF and MCPF, and then chooses the mapping that gives a higher $\Delta\mathbf{\Lambda}$.

Other Heuristics: To compare the performance of the heuristics proposed in this research (MCTF and MCPF), five other greedy heuristics were also implemented. These included: TPG, Two-Phase Greedy X (TPG-X), and two fast greedy heuristics. TPG-X is an implementation of the Max-min heuristic [16] for the environment discussed in this research. TPG-X is similar to the TPG heuristic except that in Line 4 of Figure 4, "argmax" is replaced with "argmin." The first fast greedy heuristic, denoted FGH-L, iterates through the unmapped applications in an arbitrary order, assigning an application a_i to the machine m_j such that (a) $\Delta\mathbf{\Lambda}^*(a_i, m_j) \geq 0$, and (b) $\Delta\mathbf{\Lambda}^{\text{L}}$ is maximized (ties are resolved arbitrarily). The second fast greedy heuristic, FGH-T, is similar to FGH-L except that FGH-T attempts to maximize $\Delta\mathbf{\Lambda}^{\text{T}}$.

An Upper Bound: An upper bound, UB, on the $\Delta\mathbf{\Lambda}$ value is also calculated for comparing the absolute performance of a given heuristic. The UB is equal to the $\Delta\mathbf{\Lambda}$ for a system where the following assumptions hold: (a) the communication times are zero for all applications, (b) each application a_i is mapped on the machine m_j where $\Delta\mathbf{\Lambda}_{ij}^{\text{T}}$ is maximum over all machines, and (c) that each application can use 100% of the machine where it is mapped. These assumptions are, in general, not physically realistic.

5. Simulation Experiments and Results

In this study, several sets of simulation experiments were conducted to evaluate and compare the heuristics. Experiments were performed for different values of $|\mathcal{A}|$ and $|\mathcal{M}|$, and for different types of HC environments. For all experiments, it was assumed that an application could execute on any machine.

The following simplifying assumptions were made for performing the experiments. Let n_s be the total number of sensors. The computation time function, $T_{ij}^{\text{c}}(\boldsymbol{\lambda})$, was assumed to be of the form $\sum_{1 \leq z \leq n_s} b_{ijz}\lambda_z$, where $b_{ijz} = 0$ if there is no route from the z-th sensor to application a_i. Otherwise, b_{ijz} was sampled from a Gamma distribution with a given mean and given values of "task heterogeneity" and "machine heterogeneity." (See [4] for a description of the method used in this study for generating random numbers with given mean and heterogeneity values.) The communication time functions, $T_i^{\text{t}}(\boldsymbol{\lambda})$, were similarly generated. The mean and heterogeneity parameters for communication times were kept the same as those for the computation times, because communication times in the particular target HC system [2] are of the same order as computation times.

For a given set of computation and communication time functions, the experimental set-up allowed the user to change the values of output rates and end-to-end latency constraints so as to change the "tightness" of the throughput and latency constraints. The reader is directed to [1] for details.

An experiment is characterized by the set of system parameters (e.g., $|\mathcal{A}|$, $|\mathcal{M}|$, application and machine heterogeneities) it investigates. Each experiment was repeated 90 times to obtain good estimates of the mean and standard deviation of $\Delta\mathbf{\Lambda}$. Each repetition of a given experiment will be referred to as a trial. For each new trial, a DAG with $|\mathcal{A}|$ nodes was randomly regenerated, and the values of $T_{ij}^{\text{c}}(\boldsymbol{\lambda})$ and $T_i^{\text{t}}(\boldsymbol{\lambda})$ were regenerated from their respective distributions.

Results from a typical set of experiments are shown in Figure 5. The first bar for each heuristic, titled "$\Delta\mathbf{\Lambda}^{\text{N}}$," shows the normalized $\Delta\mathbf{\Lambda}$ value averaged for all those trials in which the given heuristic successfully found a mapping. The normalized $\Delta\mathbf{\Lambda}$ for a given heuristic is equal to $\Delta\mathbf{\Lambda}$ for the mapping found by that heuristic divided by $\Delta\mathbf{\Lambda}$ for the upper bound defined in Section 4. The second bar, titled, "$\delta\boldsymbol{\lambda}^{\text{N}}$," shows the normalized $\Delta\mathbf{\Lambda}$ averaged only for those trials in which every heuristic successfully found a mapping. These figures also show, in the third bar, the value of the failure rate for each heuristic. The failure rate or FR is the ratio of the number of trials in which the heuristic could not find a mapping to the total number of trials. The interval shown at the tops of the first two bars is the 95% confidence interval [17].

Figure 5 shows the relative performance of the heuristics for the given system parameters. In this figure, FGH-T and FGH-L are not shown because of their poor failure rate and $\Delta\mathbf{\Lambda}$, respectively. It can be seen that the $\Delta\mathbf{\Lambda}$ performance difference between MCTF and MCPF is statistically insignificant. The traditional Min-min and Max-min like heuristics, i.e., TPG and TPG-X, achieve $\Delta\mathbf{\Lambda}$ values significantly lower than those for MCTF or MCPF. To make matters worse, the FR values for TPG and TPG-X are significantly higher than those for MCTF or MCPF. Even though Duplex's $\Delta\mathbf{\Lambda}$ value is statistically no better than that of MCTF or MCPF, its FR value, 12%, is about half that of MCTF or MCPF (23%).

Additional experiments were performed for various other combinations of $|\mathcal{A}|$, $|\mathcal{M}|$, and tightness of QoS constraints, and the relative behavior of the heuristics was similar to that in Figure 5. Note that all communication times

were set to zero in Figure 5 (but not in all experiments). Given the formulation of UB, it is expected that if the communication times are all zero in a given environment, then UB will be closer to the optimal value, and will make it easier to evaluate the performance of the heuristics with respect to the upper bound.

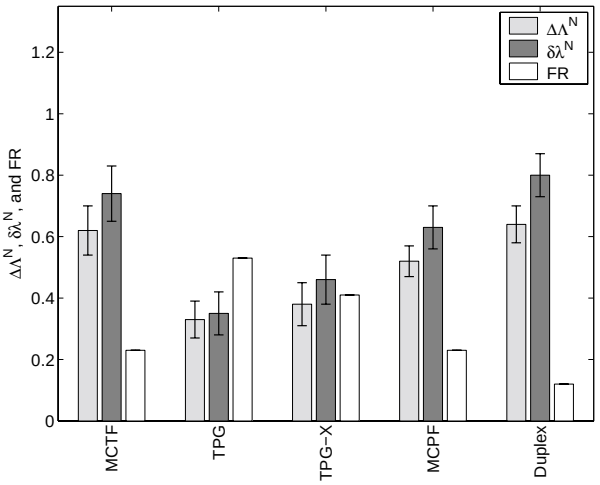

Figure 5. The relative performance of heuristics for a system where $|\mathcal{M}| = 6$, $|\mathcal{A}| = 50$. Number of sensors = number of actuators = 7. Task heterogeneity = machine heterogeneity = 0.7. All communication times were set to zero. A total of 90 trials were performed.

6. Related Work

A number of papers in the literature have studied the issue of robustness in distributed computing systems (e.g., [2, 7, 10, 11, 15, 18]). These studies are compared below with our paper.

The study in [2] is similar to the one in this paper. However, the robustness measure used in [2] makes a simplifying assumption about the way changes in $\boldsymbol{\lambda}$ can occur. Specifically, it is assumed that $\boldsymbol{\lambda}$ changes so that all components of $\boldsymbol{\lambda}$ increase in proportion to their initial values. That is, if the output from a given sensor increases by x%, then the output from all sensors increases by x%. Given this assumption, for any two sensors σ_p and σ_q, $(\lambda_p - \lambda_p^{\text{init}})/\lambda_p^{\text{init}} = (\lambda_q - \lambda_q^{\text{init}})/\lambda_q^{\text{init}} = \Delta\lambda$. For this particular definition of an increase in the system workload, any function of the vector $\boldsymbol{\lambda}$ is in reality only a function of the single scalar parameter, $\Delta\lambda$ (because $\boldsymbol{\lambda}^{\text{init}}$ is a constant vector). This research does not make this simplifying assumption; as a result, the approach taken in this paper is quite different from that in [2].

Given an allocation of a set of communicating applications to a set of machines, the work in [7] investigates the robustness of the makespan against uncertainties in the estimated execution times of the applications. The paper discusses in detail the effect of these uncertainties on the value of makespan, and how to find more robust resource allocations. Based on the model and assumptions in [7], several theorems about the properties of robustness are proven. The robustness metric in [7] was formulated for errors in the estimation of application execution times; our measure is formulated for unpredictable increases in the system load. Additionally, the formulation in [7] assumes that the execution time for any application is at most k times the estimated value, where $k \geq 1$ is the same for all applications. In our work, no such bound is assumed on the system workload.

The research in [10] considers a single-machine scheduling environment where the processing times of individual jobs are uncertain. The system performance is measured by the total flow time (i.e., the sum of *completion* times of all jobs). Given the probabilistic information about the processing time for each job, the authors determine the normal distribution that approximates the flow time associated with a given schedule. A given schedule's robustness is then given by 1 minus the risk of achieving substandard flow time performance. The risk value is calculated by using the approximate distribution of flow time. Our problem domain considers multiple machines and communication links.

The studies in [11] and [15] explore slack-based techniques for producing robust resource allocations. While [11] focusses on a job-shop environment, [15] focusses on real-time systems. The central idea is to provide each task with extra time (defined as slack) to execute so that some level of uncertainty can be absorbed without having to reallocate. However, it has been shown that when application times are known as a function of the workload, slack is not a good measure of robustness [3].

The work in [18] develops a mathematical definition for the robustness of makespan against machine breakdowns in a job-shop environment. The authors assume a certain random distribution of the machine breakdowns and a certain rescheduling policy in the event of a breakdown. Given these assumptions, the robustness of a schedule s is defined to be a weighted sum of the expected value of the makespan of the rescheduled system, M, and the expected value of the schedule delay (the difference between M and the original value of the makespan). However, the problem domain in [18] is different from ours.

7. Conclusions

Two distinct issues related to a resource allocation are investigated: its robustness and the failure rate of the heuristic used to determine the allocation. The system is expected

to operate in an uncertain environment where the workload is likely to increase unpredictably, possibly invalidating a resource allocation that was based on the initial workload estimate. The focus of this work is the design of a static heuristic that: (a) determines a maximally *robust* resource allocation, i.e., a resource allocation that maximizes the allowable increase in workload until a run-time reallocation of resources is required to avoid a QoS violation, and (b) has a very low *failure rate*. This study proposes a heuristic, called Duplex, that performs well with respect to the failure rate and the robustness towards unpredictable workload increases. Duplex was compared under a variety of simulated heterogeneous computing environments, and with a number of other heuristics taken from the literature. For all of the cases considered, Duplex gave the lowest failure rate, and a robustness value much better than that of TPG or TPG-X. Duplex is, therefore, very desirable for systems where low failure rates can be a critical requirement and where unpredictable circumstances can lead to unknown increases in the system workload.

Acknowledgments: The authors thank Sameer Shivle for his valuable comments.

References

[1] S. Ali. *Robust Resource Allocation in Dynamic Distributed Heterogeneous Computing Systems*. PhD thesis, School of Electrical and Computer Engineering, Purdue University, Aug. 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*, pages 519–530, June 2002.

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, July 2004.

[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(3):195–207, invited, Nov. 2000.

[5] 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.

[6] H. Barada, S. M. Sait, and N. Baig. Task matching and scheduling in heterogeneous systems using simulated evolution. In *10th IEEE Heterogeneous Computing Workshop (HCW 2001)* in the proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), Apr. 2001.

[7] L. Bölöni and D. C. Marinescu. Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5):395–412, Sept. 2002.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*, available at `http://www.stanford.edu/class/ee364/index.html`.

[9] 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(6):810–837, June 2001.

[10] R. L. Daniels and J. E. Carrillo. β-Robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29(11):977–985, 1997.

[11] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-based techniques for robust schedules. In *6th European Conference on Planning (ECP-2001)*, pages 7–18, Sept. 2001.

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

[13] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Transaction on Software Engineering*, SE-15(11):1427–1436, Nov. 1989.

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

[15] S. Ghosh. *Guaranteeing Fault Tolerance Through Scheduling in Real-Time Systems*. PhD thesis, Faculty of Arts and Sciences, Univ. of Pittsburgh, 1996.

[16] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

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

[18] V. J. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IEE Transactions*, 26(5):32–43, Sept. 1994.

[19] 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(2):107–131, Nov. 1999.

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

[21] 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)*, pages 375–385, May 2000.