Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems

Shoukat Ali[†], Jong-Kook Kim[†], Howard Jay Siegel[‡§], Anthony A. Maciejewski[‡], Yang Yu[*], Shriram B. Gundala[*], Sethavidth Gertphol[*], and Viktor Prasanna[*]

[†]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
{alis, kim42}@ecn.purdue.edu

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

[*]University of Southern California
Dept. of Electrical Engineering
Los Angeles, CA 90089-2560 USA
{yangyu, gundala, gertphol, prasanna}@halcyon.usc.edu

Abstract— **Recently, with the widespread use of increasingly powerful commercial off-the-shelf (COTS) products, some real-time distributed system designers have started a shift from custom-made systems to COTS-based systems to get lower costs and more flexible systems. This research investigates the problem of allocating real-time applications to a set of COTS heterogeneous machines connected together by a COTS high-speed network. For the intended distributed real-time system, the work presented in this paper includes characterizing and modeling the applications and the hardware platform, identifying and quantifying the performance goal, and designing and developing heuristics for allocating the applications so as to optimize the performance goal. Each application has certain quality of service (QoS) constraints that must not be violated (i.e., constraints on the end-to-end latency and throughput). Unlike most of the related work in real-time systems, the focus of this work is on finding an initial static allocation of the applications onto the machines to maximize the allowable increase in workload until a dynamic reallocation of resources is required to avoid a QoS violation. This paper presents and compares three greedy heuristics to solve the initial mapping problem.**

Keywords— **Real-time systems, heterogeneous computing, distributed computing, resource allocation.**

I. Introduction

Real-time systems continue to increase in importance as they are employed in various critical areas, such as command and control systems, intensive care monitoring, flight control systems, process control systems, multimedia, and high-speed communication systems. In the past, computational requirements for some real-time applications could only be met by custom-made systems. Such systems generally employed special purpose computers, interconnects, languages, and operating systems designed and built to execute those real-time applications. Typically, these custom solutions are expensive and have limited flexibility. Recently, however, with the widespread use of increasingly powerful commercial off-the-shelf (COTS) products, some real-time system designers have started a shift from custom development to COTS-based systems to get lower costs and more flexible systems [45].

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 system performance or cost-effectiveness, e.g., [11], [16], [22], [42]. A typical real-time HC system consists of heterogeneous sets of sensors, real-time applications, machines, and actuators. An important research problem is how to assign resources to applications (matching) and order the execution of the applications (scheduling) so as to maximize some performance criterion without violating any real-time constraints. This process of matching and scheduling is called mapping.

Two different types of mapping are static and dynamic. Static mapping is performed when the applications are mapped in an off-line planning phase [10], e.g., a mapping that is used when a system is first started to ensure that all real-time constraints will be met for a given initial system workload (i.e., the set of initial sensor outputs). Dynamic mapping is performed when the applications are mapped in an on-line fashion [41], e.g., when an unexpected increase in the system workload causes quality of service violations to occur [29], [52], or when new applications arrive unpredictably. In either case, the mapping problem has been shown, in general, to be NP-complete [15], [18], [30]. Thus, the development of heuristic techniques to find near-optimal mappings is an active area of research, e.g., [1], [6],

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.

[7], [10], [11], [16], [20], [41], [43], [54].

In real-time systems, static mapping approaches are very important in ensuring that all real-time constraints will be met for a given system workload *before* the system is put in operation [28], [46]. Generally, these systems operate in an environment that undergoes unexpected changes. The changes in the environment are reflected in the system parameters, e.g., system workload and the set of rates of output from different sensors. These unpredictable changes in the system may cause some QoS violation. Therefore, even though a good static mapping of real-time applications may ensure that no QoS constraints are violated when the system is first put in operation, dynamic mapping approaches may be needed to re-allocate resources at run time to avoid QoS violations, e.g., [13], [29], [31], [50], [52], [53].

This work uses a generalized performance metric that is suitable for evaluating a statically derived mapping for systems that operate in changing environments [23]. The changes in environment are reflected in the variations of run-time parameters. The general goal of this research is to delay the *first dynamic re-mapping* of resources required at run time to prevent QoS violations due to variations in the run-time parameters, specifically, the amount of the workload generated by the changing sensor outputs. This work defines the initial mapping problem as finding a static mapping of a set of applications onto a suite of machines to maximize the allowable increase in system workload until dynamic re-mapping of the applications is required to avoid a QoS violation. For the intended distributed real-time system, the contributions of this research include characterizing and modeling the applications and the hardware platform, quantifying the performance goal, designing and developing heuristics for mapping the applications so as to optimize the performance goal, and evaluating the relative performance of these heuristics.

This research, which was supported by the DARPA/ITO Quorum Program, is a continuation of a project called Management System for Heterogeneous Networks (MSHN) [25]. MSHN is a collaborative research effort among Colorado State University, Purdue University, the University of Southern California, NOEMIX, and the Naval Postgraduate School. One objective of MSHN is to design and evaluate mapping heuristics for different types of HC environments, including the COTS-based High Performance Distributed Computing Program (HiPer-D) environment at the Naval Surface Warfare Center, Dahlgren Division (NSWCDD) [45]. A general framework for a HiPer-D subsystem is shown in Figure 1 and a specific example is shown in Figure 2.

The remainder of this paper is organized in the following manner. Some representative related work is given in Section II. Section III develops models for the applications and the hardware platform. These models are then used in Section IV to derive expressions for computation and communication times. Section V presents an quantitative measure of the performance goal for evaluating the quality of a given mapping of applications to machines. Three greedy heuristics to solve the initial mapping problem are described in Section VI, along with the experimental setup and the results for the evaluation of these heuristics.

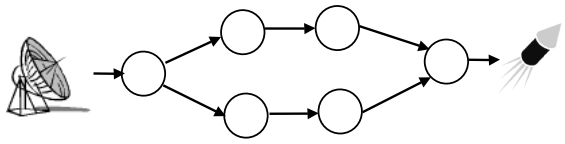

Fig. 1. An example real-time system where the output from a radar is processed by a group of applications, and a signal is then produced to control the firing of a missile. The nodes denote applications in the real-time system.

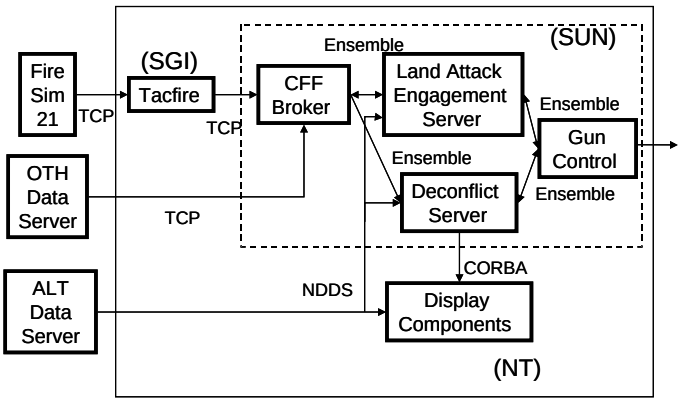

Fig. 2. An example HiPer-D subsystem composed of heterogeneous COTS machines and networks. Fire Sim 21, OTH (Over the Horizon) Data Server, and ALT (Air Engagement Control Local Track) Data Server send periodic data to the applications. Tacfire, CFF (Call for Fire) Broker, Land Attack Engagement Server, Deconflict Server, Gun Sim, 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.

II. Related Work

Many research efforts in the literature concentrate on scheduling real-time applications on a processor (e.g., [5], [8], [14], [24], [27], [33], [34], [35], [36], [37], [38], [39], [44], [49], [55], [56]). The major ways in which our work differs from the works cited above are: (a) the system environment of our research is very different from most of the previous research (e.g., multitasking heterogeneous distributed machines capable of round-robin scheduling only, and executing communicating applications), (b) our research focuses on the allocation aspect of the mapping problem, while most of the literature cited above focuses on the post-assignment part of the mapping problem, i.e., the scheduling of applications on a machine with an assumed allocation or a simple allocation scheme, and (c) we use a very different performance metric that measures how well-prepared our static mapping is for absorbing run-time variations in the system load. The purpose of our metric is to evaluate a statically derived mapping for systems that operate in dynamic environments.

Research efforts in (e.g., [9], [12], [17], [26], [46], [47]) do consider task assignment as well as task scheduling.

The research in [9] and [12] used a very simple partition of processors for allocation of tasks to each partition. In the research efforts of [26] and [46], a branch and bound search algorithm was used for the allocation phase. Even though these efforts had allocation schemes, these methods were not directly suited for our research because our environment is very different, consisting of a heterogeneous suite of machines with multitasking multiple processors and communication links using a round robin scheduling policy. Additionally our performance metric focuses on evaluating a statically derived mapping for systems that operate in dynamic environments.

The mapping approaches in [17] and [47] allocate and schedule tasks, but the system environment they assume is different. The research effort in [47] assumes that applications are independent with no precedence constraints, no communication among applications, no multitasking processors and communication links, and run-time variations are not considered. The research in [17] assumes a homogeneous multiprocessor system and concentrates on the scheduling aspect of the mapping problem. The fact that processors and communication links can perform multitasking makes the problem of mapping applications to machines very different. When applications are mapped to processors, the processing time of a application depends on other applications mapped onto the same machine. Even when communication is considered in the other research efforts mentioned, it is considered only as a constant delay or the delay is independent of other communications. However, if the links can be multi-tasked, the communication delay varies depending on the mapping itself. Invoking a dynamic resource allocation/scheduler to manage the variation in run-time variables can be costly in terms of resources used and jobs not completed before a certain time constraint. Therefore an analysis of the impact of the changing workload on application execution and communication requirements for the purpose of delaying the invocation of the dynamic scheduler in the target system is one of the contributions of this research.

The system model assumed in our research is similar to that of the DeSiDeRaTa project (e.g., [13], [29], [50], [51], [52], [53]) in that both use continuously running applications and a heterogeneous distributed system. The difference is that, while our work focuses on deriving a static mapping of the applications to resources, the DeSiDeRaTa project focuses on dynamically re-mapping the resources to meet the real-time constraints. Our work concentrates on producing an initial static mapping that can sustain a maximum allowable increase in load before dynamic resource allocation procedures developed by the DeSiDeRaTa project have to be invoked.

III. System Model

The system considered here consists of heterogeneous sets of sensors, real-time applications, machines, and actuators. 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 can be modeled with a directed acyclic graph (DAG) consisting of a set $\mathcal{A}$ of nodes. The nodes in the DAG correspond to the applications and the arcs between the nodes represent the data transfers between applications. Figure 3 shows a DAG representation of the applications and the data transfers, along with the sensors and actuators in the system. (The dashed lines in Figure 3 can be ignored for now.) Let a_i be the i-th application in $\mathcal{A}$ (numbered arbitrarily).

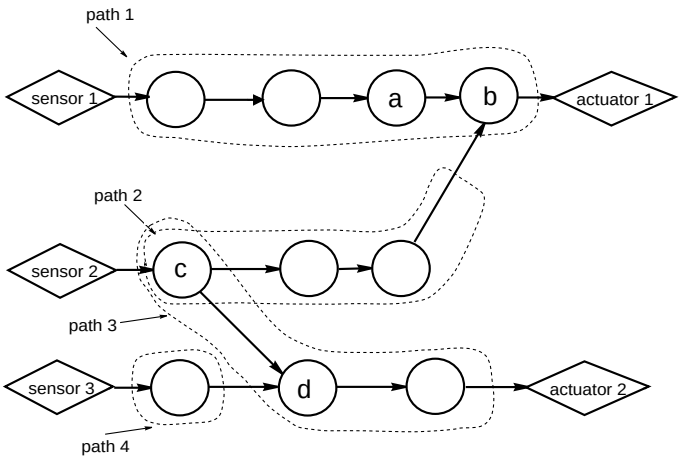

Fig. 3. The DAG model for the applications and data transfers. The dashed lines enclose each path formed by the applications.

It is assumed that, for a given data set, the precedent-constrained applications use "greedy synchronization" [40] to schedule their executions. Specifically, a successor application starts processing a particular data set as soon as that data set is received. Unlike many other real-time applications (e.g., in [46], [28]), the applications in this system are, in general, not periodic in the sense that they are not invoked periodically. It is also assumed that an application starts transferring data only after it has completed processing the current input.

This research assumes that all multiple input applications are either combining or discriminating applications. A "combining" application produces an output by combining or using all of its inputs. The output is produced at the rate of the slowest input. Another type of a multiple-input application is a "discriminating" application, which does not produce an output unless a particular input stream (called the trigger input) supplies a new data set. The output is produced at the rate of the trigger input. Because a combining application can be modeled as a discriminating application with the trigger input being the slowest rate input, this research considers only discriminating applications.

Typically, the data stream from a sensor is successively processed by a sequence of applications, and the output from the last application serves as an input to an actuator or to a discriminating application. Such a chain of

producer-consumer pairs forms an "end-to-end computation" [48], and is referred to as a path in this research (as well as in [50], [53]). Often there is a deadline associated with the end-to-end computation performed by a path. In Figure 3, paths are shown by the dashed lines. The trigger inputs for applications b and d come from applications a and c, respectively. Let $\mathcal{P}$ be the set of paths formed by applications in $\mathcal{A}$. Let $\mathcal{P}_k$ be the list of applications that comprise the k-th path. Let $\alpha_{k,i}$ be the i-th application in $\mathcal{P}_k$, where $1 \leq i \leq |\mathcal{P}_k|$ ($|\mathcal{P}_k|$ is the total number of applications in path $\mathcal{P}_k$). The applications in $\mathcal{P}_k$ are numbered sequentially. Note that an application may be present in multiple paths, i.e., it is possible that $\alpha_{k,i} = \alpha_{l,j} = a_h$. Similarly, a given data transfer may be a part of multiple paths.

The sensors constitute the interface of the system to the external world for retrieving information. These sensors may be radars, sonars, cameras monitoring a conveyor belt, etc. A sensor is characterized by (a) the maximum rate at which it produces data sets, and (b) the amount of load a given data set presents to the applications that have to process the data set. Typically, different sensors have different rates at which they produce data sets. Also, the measure of load may be different for different sensors. Let σ_z be the z-th sensor in the set of sensors, $\boldsymbol{\sigma}$. Let R_z be the maximum periodic data output rate from σ_z. In this study, the load measure, λ_z, is defined to be the number of objects present in one data set from the sensor σ_z in the most recent cycle. The system workload, $\boldsymbol{\lambda}$, is a measure of the load values from all sensors, and is the vector,

$$\boldsymbol{\lambda} = \left[\lambda_1 \cdots \lambda_{|\boldsymbol{\sigma}|}\right]^{\mathrm{T}}.$$

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 $\boldsymbol{\lambda}^{\text{init}}$, i.e.,

$$\boldsymbol{\lambda}^{\text{init}} = \left[\lambda_1^{\text{init}} \cdots \lambda_{|\boldsymbol{\sigma}|}^{\text{init}}\right]^{\mathrm{T}}.$$

The system also contains a set $\mathcal{M}$ of heterogeneous COTS machines. The machines have COTS operating systems. Specifically, it is assumed that when multiple applications are computed on a machine, the operating system uses a round-robin scheduling policy to allocate the CPU to the applications. Once an application finishes processing its data, the application is suspended until the next set of input data arrives for this application. It is assumed that an interrupt is used to put the application back on the list of applications actively using the CPU. Similarly, when there are multiple data transfers originating at a machine, it is assumed that the operating system uses a round-robin scheduling policy to allocate the network link to the data transfers. Section IV develops expressions for the computation and communication times using the above assumptions.

All machines are connected by heterogeneous dedicated full-duplex communication links to a non-blocking switch. The sensors, as well as actuators, are connected by half-duplex connections to the same non-blocking switch.

Several QoS constraints, e.g., maximum end-to-end latency, inter-processing time, and throughput, have been discussed in [53]. In this research, the maximum end-to-end latency and throughput constraints will be investigated, as they are most pertinent to the problem domain. The maximum end-to-end latency constraint states that, for a given path $\mathcal{P}_k$, the time taken between the instant the sensor feeding the path outputs a data set to the instant the actuator or the discriminating application fed by the path receives the result of the computation on that data set must be no greater than a given value, $L_{\max}(\mathcal{P}_k)$. (Not every input to an actuator activates it.)

Also, for each path $\mathcal{P}_k$, the input data rate, $\rho(\mathcal{P}_k)$, is defined to be the output data rate of the sensor that feeds $\mathcal{P}_k$. 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 $1/\rho(\mathcal{P}_k)$. Let $r(a_i)$ be defined as $\rho(\mathcal{P}_k)$ where $a_i \in \mathcal{P}_k$.

IV. Computation and Communication Models

A. Computation Model

For an application, the estimated time to compute a given data set depends on the load presented by the data set, and the machine executing the application. Let $\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$ be the estimated time to compute for application a_i on m_j for a given data set when a_i is the only application executing on m_j. This research assumes that $\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$ is a function known for all i, j, and $\lambda_z \in \boldsymbol{\lambda}$.

The effect of multitasking on the computation time of an application is accounted for by assuming that all applications mapped to a machine are processing data continuously. Let $N(m_j)$ be the number of applications executing on machine m_j. Let $T^{\mathrm{c}}(a_i, m_j, \boldsymbol{\lambda})$ be the computation time for a_i on machine m_j when a_i shares this machine with other applications. If the overhead due to context switching is ignored, $T^{\mathrm{c}}(a_i, m_j, \boldsymbol{\lambda})$ will be $N(m_j) \times \mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$.

However, the overhead in computation time introduced by context switching is not trivial in a round-robin scheduler. A simple expression for the context switching overhead is now developed. The context switching overhead that an application incurs over the course of its execution on a given machine depends on the ETC value of the application on the machine, and the number of applications executing on the machine. Let $O_{\mathrm{cs}}(a_i, m_j, \boldsymbol{\lambda})$ be the context switching overhead incurred when application a_i executes a given data set on machine m_j. Let $T_{\mathrm{cs}}(m_j)$ be the time m_j needs in switching the execution from one application to another. Let $T_q(m_j)$ be the size (quantum) of the time slice given by m_j to each application in the round-robin scheduling policy used on m_j. Then,

$$O_{\mathrm{cs}}(a_i, m_j, \boldsymbol{\lambda}) = \begin{cases} 0 & \text{if } N(m_j) = 1 \\ \dfrac{\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda}) \times N(m_j)}{T_q(m_j)} \times T_{\mathrm{cs}}(m_j) & \text{if } N(m_j) > 1 \end{cases}$$

$T^{\mathrm{c}}(a_i, m_j, \boldsymbol{\lambda})$ can now be stated as,

$$T^{\mathrm{c}}(a_i, m_j, \boldsymbol{\lambda}) = \begin{cases} \mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda}) & \text{if } N(m_j) = 1 \\ \mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda}) \times N(m_j) \left(1 + \frac{T_{\mathrm{cs}}(m_j)}{T_q(m_j)}\right) & \text{if } N(m_j) > 1 \end{cases}$$

B. Communication Model

This section develops an expression for the time needed to transfer the output data from a source application to a destination application at a given load. Let $M(a_i, a_j, \boldsymbol{\lambda})$ be the size of the message data sent from application a_i to a destination application a_j at the given load. Let $\mu(a_i)$ be the machine on which a_i is mapped. Let $T^{\mathrm{n}}(a_i, a_j, \boldsymbol{\lambda})$ be the time to send data from application a_i to application a_j at the given load.

A model for calculating the communication times should identify the various steps involved in effecting a data transfer. The two steps identified in [19] for communication in a similar domain are the communication setup time and the queuing delay. These steps contribute to the overall communication time to different extents depending on the intended communications environment.

Communication Setup Time. Before data can be transferred from one machine to another, a communication setup time is required for setting up a *logical* communication channel between the sender application and the destination application. Once established, a logical communication channel is torn down only when the sending or receiving application is finished executing. Because this study considers continuously executing applications (because the sensors continually produce data), the setup time will be incurred only once, and will, therefore, be amortized over the course of the application execution. Hence, the communication setup time is ignored in this study.

Queuing Delay. A data packet is queued twice enroute from the source machine to the destination machine. First, the data packet is queued in the output buffer of the source machine, where it waits to use the communication link from the source machine to the switch. The second time, the data packet is queued in the output port of the switch, where it waits to use the communication link from the switch to the destination machine. Note that the switch has a separate output port for each machine in the system.

The queuing delay at the sending machine is modeled by assuming that the bandwidth of the link from the sending machine to the switch is shared equally among all data transfers originating at the sending machine. This will underestimate the bandwidth available for each transfer because it assumes that all of the other transfers are always being performed. Similarly, the queuing delay at the switch is modeled by assuming that the bandwidth of the link from the switch to the destination machine is equally divided among all data transfers originating at the switch and destined for the destination machine. Let $B(\mu(a_i), \mathrm{swt})$ be the bandwidth (in bytes per unit time) of the communication link between $\mu(a_i)$ and the switch, and $B(\mathrm{swt}, \mu(a_j))$ be the bandwidth (in bytes per unit time) of the communication link between the switch and $\mu(a_j)$. Let $N_{\mathrm{ct}}(\mu(a_i), \mathrm{swt})$ be the number of data transfers using the communication link from $\mu(a_i)$ to the switch. The subscript "ct" stands for "contention." Let $N_{\mathrm{ct}}(\mathrm{swt}, \mu(a_j))$ be the number of data transfers using the communication link from the switch to $\mu(a_j)$. Then, $T^{\mathrm{n}}(a_i, a_j, \boldsymbol{\lambda})$, the time to transfer the output data from application a_i to a_j, is given by:

$$T^{\mathrm{n}}(a_i, a_j, \boldsymbol{\lambda}) = M(a_i, a_j, \boldsymbol{\lambda}) \times \left(\frac{N_{\mathrm{ct}}(\mu(a_i), \mathrm{swt})}{B(\mu(a_i), \mathrm{swt})} + \frac{N_{\mathrm{ct}}(\mathrm{swt}, \mu(a_j))}{B(\mathrm{swt}, \mu(a_j))}\right) \quad (1)$$

The above expression can also accommodate the situations when a sensor communicates with the first application in a path, or when the last application in a path communicates with an actuator. The sensor feeding $\mathcal{P}_k$ can be treated as a "pseudo-application" that has a zero computation time and is already mapped to an imaginary machine, and, as such, can be denoted by $\alpha_{k,0}$. Similarly, the actuator receiving data from $\mathcal{P}_k$ can also be treated as a pseudo-application with a zero computation time, and will be denoted by $\alpha_{k,|\mathcal{P}_k|+1}$. Accordingly, for these cases: $M(\alpha_{k,0}, \alpha_{k,1}, \boldsymbol{\lambda})$ corresponds to the load from the sensor; $B(\mu(\alpha_{k,0}), \mathrm{swt})$ is the bandwidth of the link between the sensor and the switch; and $B(\mathrm{swt}, \mu(\alpha_{k,|\mathcal{P}_k|+1}))$ corresponds to the bandwidth of the link between the switch and the actuator. Similarly, $N_{\mathrm{ct}}(\mu(\alpha_{k,0}), \mathrm{swt})$ and $N_{\mathrm{ct}}(\mathrm{swt}, \alpha_{k,|\mathcal{P}_k|+1}))$ both are 1. In the situation where $\mu(a_i) = \mu(a_j)$, one can interpret $N_{\mathrm{ct}}(\mu(a_i), \mathrm{swt})$ and $N_{\mathrm{ct}}(\mathrm{swt}, \mu(a_j))$ both as 0; $T^{\mathrm{n}}(a_i, a_j, \boldsymbol{\lambda}) = 0$ in that case.

Note that, analogous to $\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$, $\mathrm{ETK}(a_i, a_j, m_p, m_q, \boldsymbol{\lambda})$ can be defined as the estimated time to communicate for application a_i with application a_j when a_i is the only application executing on m_p, and a_j is the only application executing on m_q.

$$\mathrm{ETK}(a_i, a_j, m_p, m_q, \boldsymbol{\lambda}) = M(a_i, a_j, \boldsymbol{\lambda}) \times \left(\frac{1}{B(m_p, \mathrm{swt})} + \frac{1}{B(\mathrm{swt}, m_q)}\right)$$

V. Performance Goal

A. The Quantitative Statement

Recall from Section I that the initial mapping problem consists of finding an initial static allocation of applications onto machines to maximize the allowable increase in system workload, $\boldsymbol{\lambda}$, until dynamic reallocation of resources is required to avoid a QoS violation. Recall that the outputs of all sensors in the system constitute $\boldsymbol{\lambda}$. The increase 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. This research makes the simplifying assumption that outputs from all sensors increase at the

same rate. 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, $(\lambda_p - \lambda_p^{\text{init}})/\lambda_p^{\text{init}} = (\lambda_q - \lambda_q^{\text{init}})/\lambda_q^{\text{init}} = \Delta\lambda$. $\Delta\lambda$ will be referred to as the percentage increase in the system load. Now, $\boldsymbol{\lambda}$ could be re-expressed in terms of $\Delta\lambda$ as

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_{|\boldsymbol{\sigma}|} \end{bmatrix} = \begin{bmatrix} (1+\Delta\lambda) \times \lambda_1^{\text{init}} \\ \vdots \\ (1+\Delta\lambda) \times \lambda_{|\boldsymbol{\sigma}|}^{\text{init}} \end{bmatrix} = (1+\Delta\lambda) \times \boldsymbol{\lambda}^{\text{init}}.$$

Clearly, there are many other ways of defining an increase in the system workload. The definition chosen here reflects the load characteristics for one particular kind of HiPer-D system. Note that 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).

The rest of this section develops a quantitative measure for the performance goal. Let $\Delta\Lambda^{\text{L}}(\mathcal{P}_k)$ be the maximum $\Delta\lambda$ that can be processed by path $\mathcal{P}_k$ without violating its end-to-end latency constraint. (The "L" in the superscript denotes "latency.") Let $\Delta\Lambda^{\text{L, SW}}$ be the maximum $\Delta\lambda$ that can be processed by the system without violating the end-to-end latency constraint for any path. (The "SW" in the superscript denotes "system-wide.") Clearly,

$$\Delta\Lambda^{\text{L, SW}} = \min_{\mathcal{P}_k \in \mathcal{P}} \left(\Delta\Lambda^{\text{L}}(\mathcal{P}_k) \right). \tag{2}$$

Note that if $\Delta\Lambda^{\text{L, SW}} < 0$ for a certain allocation of applications to machines, then it means that some path in the system is violating its latency constraint at the initial system workload.

Let $\Delta\Lambda^{\text{T, C}}(a_i, \mu(a_i))$ be the maximum $\Delta\lambda$ that the CPU on machine $\mu(a_i)$ can process for application a_i without a throughput violation. (The "T" in the superscript denotes "throughput," and the "C" denotes "CPU.") Similarly, let $\Delta\Lambda^{\text{T, N}}(a_i, a_j)$ be the maximum $\Delta\lambda$ that the communication link between a_i and a_j can process without a throughput violation. (The "N" in the superscript denotes "network.") Let $\mathcal{D}(a_i)$ be the set of successor applications of a_i. In addition, let $\Delta\Lambda^{\text{T}}(a_i, \mu(a_i))$ be the minimum of $\Delta\Lambda^{\text{T, C}}(a_i, \mu(a_i))$ and $\min_{a_j \in \mathcal{D}(a_i)}(\Delta\Lambda^{\text{T, N}}(a_i, a_j))$. If $\Delta\Lambda^{\text{T, SW}}$ is the maximum $\Delta\lambda$ that can be processed by the system without violating the throughput constraint for any application or data transfer, then,

$$\Delta\Lambda^{\text{T, SW}} = \min_{a_i \in \mathcal{A}} \left(\Delta\Lambda^{\text{T}}(a_i, \mu(a_i)) \right). \tag{3}$$

Note that if $\Delta\Lambda^{\text{T, SW}} < 0$ for a certain allocation of applications to machines, then it means that some application in the system is violating its throughput constraint at the initial system workload.

The goal is to find a mapping that maximizes the system-wide maximum tolerable percentage increase in load, $\Delta\Lambda^{\text{SW}}$, without violating any QoS constraint. The value of $\Delta\Lambda^{\text{SW}}$ is given by $\min(\Delta\Lambda^{\text{L, SW}}, \Delta\Lambda^{\text{T, SW}})$. Note that if $\Delta\Lambda^{\text{SW}} < 0$ for a certain allocation of applications to machines, then it means that either a throughput or a latency constraint is being violated. Therefore, deriving a mapping that ensures $\Delta\Lambda^{\text{SW}} \geq 0$ ensures that all QoS constraints will be met when the applications are allocated according to that mapping.

If $\Delta\Lambda^{\text{SW}}$ for all optimal mappings equals $\Delta\Lambda^{\text{T, SW}}$, the system is said to be throughput-constrained. A latency-constrained system is defined as a system where $\Delta\Lambda^{\text{SW}}$ for all optimal mappings equals $\Delta\Lambda^{\text{L, SW}}$.

B. Satisfying the Throughput Constraint

To ensure that the throughput constraint for application a_i or a data transfer from a_i to a successor application a_j is not violated,

$$T^{\text{c}}(a_i, m_j, \boldsymbol{\lambda}) \leq 1/r(a_i), \tag{4}$$

and

$$T^{\text{n}}(a_i, a_j, \boldsymbol{\lambda}) \leq 1/r(a_i). \tag{5}$$

The left hand sides (LHS s) of inequalities 4 and 5 can be expressed as functions of $\Delta\lambda$. The value of $\Delta\lambda$ for which the LHS in inequality 4 equals the right hand side (RHS) is equal to $\Delta\Lambda^{\text{T, C}}(a_i, \mu(a_i))$. Similarly, the value of $\Delta\lambda$ for which the LHS in inequality 5 equals the RHS is equal to $\Delta\Lambda^{\text{T, N}}(a_i, a_j)$. After finding $\Delta\Lambda^{\text{T, C}}(a_i, \mu(a_i))$ and $\Delta\Lambda^{\text{T, N}}(a_i, a_j)$ for all applications, Equation 3 can be used to determine $\Delta\Lambda^{\text{T, SW}}$.

C. Satisfying the Latency Constraint

To ensure that the latency constraint for a given path is not violated, the actual end-to-end latency must not be larger than $L_{\max}(\mathcal{P}_k)$. The above constraint can now be defined mathematically as,

$$\sum_{i=1}^{|\mathcal{P}_k|} T^{\text{c}}(\alpha_{k,i}, \mu(\alpha_{k,i}), \boldsymbol{\lambda}) + \sum_{i=0}^{|\mathcal{P}_k|} T^{\text{n}}(\alpha_{k,i}, \alpha_{k,i+1}, \boldsymbol{\lambda}) \leq L_{\max}(\mathcal{P}_k) \tag{6}$$

The LHS of inequality 6 can be expressed as a function of $\Delta\lambda$. The value of $\Delta\lambda$ for which the LHS equals the RHS is equal to $\Delta\Lambda^{\text{L}}(\mathcal{P}_k)$. One can determine $\Delta\Lambda^{\text{L}}(\mathcal{P}_k)$ for all paths, and then use Equation 2 to find $\Delta\Lambda^{\text{L, SW}}$.

VI. Heuristic Descriptions

This section develops three greedy heuristics for the initial mapping problem. Greedy techniques perform well in many situations, and have been well-studied (e.g., [30], [4]). Two of these three heuristics, the Most Critical Task First (MCTF) and Tie-Breaking Two Phase Greedy (TB TPG) heuristics, are designed to work well in throughput-constrained real-time heterogeneous systems. The other heuristic, the Most Critical Path First (MCPF) heuristic, is designed to work well in latency-constrained real-time heterogeneous systems.

It is important to note that these heuristics use the $\Delta\Lambda^{\text{SW}}$ value to guide the heuristic search; however, the procedure given in Section V for calculating $\Delta\Lambda^{\text{SW}}$ assumes

that a complete mapping of all applications is known. But during the course of the execution of the heuristics, not all applications are mapped. So, for the purpose of these heuristics, the calculation of $\Delta\Lambda^{\mathrm{SW}}$ ignores the unmapped applications. This study also investigated the possibility of using average values for the computation and communication times of the unmapped applications, and concluded that using average values gives significant reductions in performance for the heuristics discussed here. This is discussed further in [2].

The MCTF heuristic makes one application to machine assignment in each iteration. Each iteration can be split into two phases. Let $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j)$ be the value of $\Delta\Lambda^{\mathrm{SW}}$ *if* application a_i is mapped on m_j. Similarly, let $\Delta\Lambda^{\mathrm{T}^*}(a_i, m_j)$ be the value of $\Delta\Lambda^{\mathrm{T}}(a_i, m_j)$ *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

$$m_j = \underset{m_k \in \mathcal{M}}{\operatorname{argmax}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_k)). \tag{7}$$

(Note that $\operatorname{argmax}_x f(x)$ returns the value of x that maximizes the function $f(x)$. If there are multiple values of x that maximize $f(x)$, then $\operatorname{argmax}_x f(x)$ returns the set of all those values.) If the RHS in Equation 7 returns a set of machines, $G(a_i)$, instead of a unique machine, then $m_j = \operatorname{argmax}_{m_k \in G(a_i)}(\Delta\Lambda^{\mathrm{T}^*}(a_i, m_k))$, i.e., the individual throughput constraints are used to break ties in the overall system-wide measure. Note that 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 $\Delta\Lambda_g^{\mathrm{SW}}$ be the value of $\Delta\Lambda^{\mathrm{SW}}$ 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

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

The application a_x is then assigned to the machine m_y. Assigning any other application makes the value of $\Delta\Lambda_1^{\mathrm{SW}}$ higher than what the final $\Delta\Lambda^{\mathrm{SW}}$, i.e., $\Delta\Lambda_{|\mathcal{A}|}^{\mathrm{SW}}$, can possibly be. This is likely to "misguide" the heuristic search in the later iterations. (Note that the discussion above does not imply that an optimal mapping must contain the assignment of a_x on m_y.)

The criterion used to make the second phase application to machine assignment for iteration number 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 $\Delta\Lambda^{\mathrm{SW}}$. Let $\mathcal{A}_{\mathrm{map}}$ be the set of all currently mapped applications. 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 $\Delta\Lambda^{\mathrm{SW}^*}(a_i,\ m_x^{a_i}) \geq \Delta\Lambda^{\mathrm{SW}^*}(a_i,\ m_y^{a_i})$ if $x < y$. Let v be an integer such that $2 \leq v \leq |\mathcal{M}|$, and let $C(a_i, v)$ be the percentage decrease in $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j)$ if a_i is mapped on $m_v^{a_i}$ (its v-th best machine) instead of $m_1^{a_i}$, i.e.,

$$C(a_i, v) = \frac{\Delta\Lambda^{\mathrm{SW}^*}(a_i,\ m_1^{a_i}) - \Delta\Lambda^{\mathrm{SW}^*}(a_i,\ m_v^{a_i})}{\Delta\Lambda^{\mathrm{SW}^*}(a_i,\ m_1^{a_i})}.$$

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

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

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 4. The technique shown in Figure 4 builds on the idea of the Sufferage heuristic given in [41].

```
(1)  initialize: v = 2; F = the set of (a_i, m_j) pairs
     from the first phase
(2)  for v = 2 to |M|
(3)      if argmax_{(a_i,m_j) ∈ F}(C(a_i, v)) is a unique
         pair (a_x, m_y)
(4)          return (a_x, m_y)
(5)      else
(6)          F = the set of pairs returned by
             argmax_{(a_i,m_j) ∈ F}(C(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. */
     /* F is now the set of (a_i, m_j) pairs from */
     /* the last execution of Line 6*/
(8)  if argmax_{(a_i,m_j) ∈ 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 argmax_{(a_i,m_j) ∈ F}(T(a_i, 2))
```

Fig. 4. Selecting the most critical application to map next given the set of (a_i, m_j) pairs from the first phase of MCTF.

This research also proposes a modified version of the Min-min heuristic for mapping in throughput-constrained systems. Variants of the Min-min heuristic (first presented in [30]) have been studied, e.g., [4], [10], [21], [41], [54], 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 5, and the modified Min-min, called the Tie-Breaker Two-Phase Greedy (TB TPG) is shown in Figure 6. TB TPG augments TPG with two features borrowed from MCTF: (a) the special first iteration, and (b) the use of application criticality to resolve any ties in Line 9 of Figure 6.

(1) **do** until all applications are mapped
(2) **for** each unmapped application a_i, find the machine m_j such that $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j) = \max_{m_k \in \mathcal{M}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_k))$; resolve ties arbitrarily
(3) **if** $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j) < 0$, 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 $\Delta\Lambda^{\mathrm{SW}^*}(a_x, m_y) = \max_{(a_i,m_j)\text{ pairs}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j))$; resolve ties arbitrarily
(5) map a_x on m_y
(6) **enddo**

Fig. 5. The TPG heuristic.

(1) flag = 0
(2) **do** until all applications are mapped
(3) **for** each unmapped application a_i, find the machine m_j such that $G(a_i) = \mathrm{argmax}_{m_k \in \mathcal{M}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_k))$, and $m_j = \mathrm{argmax}_{m_k \in G(a_i)}(\Delta\Lambda^{\mathrm{T}^*}(a_i, m_k))$
(4) **if** $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j) < 0$, this heuristic cannot find a mapping
(5) **if** flag = 0
/* special case for the first iteration */
(6) from (a_i, m_j) pairs found above, select the pair (a_x, m_y) such that $\Delta\Lambda^{\mathrm{SW}^*}(a_x, m_y) = \min_{(a_i,m_j)\text{ pairs}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j))$; resolve ties arbitrarily
(7) flag = 1
(8) **else**
/* for all but the first iteration */
(9) from (a_i, m_j) pairs found above, select the pair(s) (a_x, m_y) such that $\Delta\Lambda^{\mathrm{SW}^*}(a_x, m_y) = \max_{(a_i,m_j)\text{ pairs}}(\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j))$; resolve any ties by choosing the most critical application first
(10) map a_x on m_y
(11) **enddo**

Fig. 6. The TB TPG heuristic.

The MCPF heuristic explicitly considers the latency constraints of the paths in the system, and is designed to work well in latency-constrained systems. MCPF 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 the ranked order. The modified form of MCTF differs from MCTF in that the first iteration is the same as the subsequent iterations.

The ranking procedure used by MCPF is now explained in detail. Let $L_{\mathrm{best}}(\mathcal{P}_k, \Delta\lambda)$ be a lower bound on the value of the end-to-end latency for the path $\mathcal{P}_k$ when $\mathcal{P}_k$ is processing a load increase of $\Delta\lambda$.

$$L_{\mathrm{best}}(\mathcal{P}_k, \Delta\lambda) = \sum_{\alpha_{k,i} \in \mathcal{P}_k} \min_{m_j \in \mathcal{M}} \big(\mathrm{ETC}(\alpha_{k,i}, m_j, \boldsymbol{\lambda})\big).$$

The lower bound ignores communications and assumes that every application can use 100% of the machine on which it is executing.

Let slack, $S(\mathcal{P}_k, \Delta\lambda)$, for a given path $\mathcal{P}_k$ be defined as

$$S(\mathcal{P}_k, \Delta\lambda) = \frac{L_{\max}(\mathcal{P}_k) - L_{\mathrm{best}}(\mathcal{P}_k, \Delta\lambda)}{L_{\max}(\mathcal{P}_k)}.$$

The heuristic ranks the paths in an ordered list $G = \langle \mathcal{P}_1^{\mathrm{crit}}, \mathcal{P}_2^{\mathrm{crit}}, \cdots, \mathcal{P}_{|\mathcal{P}|}^{\mathrm{crit}} \rangle$ such that $S(\mathcal{P}_x^{\mathrm{crit}}, \Delta\lambda) \leq S(\mathcal{P}_y^{\mathrm{crit}}, \Delta\lambda)$ if $x < y$. The slack for a given path depends on the load at which it is currently operating. Therefore, the order in the list G is a function of the $\Delta\lambda$ value at which the $S(\mathcal{P}_k, \Delta\lambda)$ values for different paths are evaluated. Ideally, one should derive the path rankings at the value of $\Delta\lambda$ where $S(\mathcal{P}_1^{\mathrm{crit}}, \Delta\lambda) = 0$. A binary search procedure for determining such a value of $\Delta\lambda$ is given in [2].

For an arbitrary HC system, one is not expected to know if the system is latency-constrained or throughput-constrained, or neither. In that case, this research proposes running both MCTF (or TB TPG) 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 $\Delta\Lambda^{\mathrm{SW}}$.

To compare the performance of the heuristics proposed in this research (MCTF, TB TPG, and MCPF), five other greedy heuristics were also implemented. These included: TPG, Two-Phase Greedy X (TPG-X), Tie-Breaker Two-Phase Greedy X (TB TPG-X), and two fast greedy heuristics. TPG-X is an implementation of the Max-min heuristic [30] for the environment discussed in this research. TPG-X is similar to the TPG heuristic except that in Line 4 of Figure 5, "max" is replaced with "min." TB TPG-X is related to TB TPG in the same way TPG-X is related to TPG. 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) $\Delta\Lambda^{\mathrm{SW}^*}(a_i, m_j) \geq 0$, and (b) $\Delta\Lambda^{\mathrm{L,\,SW}}$ 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 $\Delta\Lambda^{\mathrm{T,\,SW}}$.

VII. 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.

For the experiments presented here, $\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$ functions were generated in the following manner. Let f_{ijz} be a linear function of λ_z, with a slope $s^{\mathrm{c}}(a_i, m_j, \lambda_z)$ and an intercept $i^{\mathrm{c}}(a_i, m_j, \lambda_z)$. Then $\mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})$ was set to be the sum of all f_{ijz} such that there is a route from the sensor σ_z to a_i. Similarly, for generating functions for $M(a_i, a_j, \boldsymbol{\lambda})$), let g_{ijz} be a linear function of λ_z, with a slope $s^{\mathrm{n}}(a_i, a_j, \lambda_z)$ and an intercept $i^{\mathrm{n}}(a_i, a_j, \lambda_z)$. Then, $M(a_i, a_j, \boldsymbol{\lambda})$) was set to be the sum of all g_{ijz} such that there is a route from the sensor σ_z to a_i. These assumptions reflect one particular type of HC system. *However, the model developed in this study does not assume or require ETC$(a_i, m_j, \boldsymbol{\lambda})$ or M$(a_i, a_j, \boldsymbol{\lambda})$ to be linear functions of $\boldsymbol{\lambda}$. Any non-decreasing function can be used.*

For each application (a_i), machine (m_j), and sensor ($\lambda_z \in \boldsymbol{\lambda}$), the $s^{\mathrm{c}}(a_i, m_j, \lambda_z)$ and $i^{\mathrm{c}}(a_i, m_j, \lambda_z)$ values were generated by randomly sampling a Gamma distribution. The mean was arbitrarily set to twelve, the "task heterogeneity" was set to 0.4, and the "machine heterogeneity" was also set to 0.4 (heterogeneity is the standard deviation divided by the mean). See [3] for a description of the method used in this study for generating random numbers with given mean and heterogeneity values. The chosen values are characteristic of systems with high application and machine heterogeneities [3]. Similarly, for each a_i, a_j, and λ_z, the $s^{\mathrm{n}}(a_i, a_j, \lambda_z)$ and $i^{\mathrm{n}}(a_i, a_j, \lambda_z)$ values were also sampled from a Gamma distribution. The parameters of this distribution were kept the same as those for generating the $s^{\mathrm{c}}(a_i, m_j, \lambda_z)$ and $i^{\mathrm{c}}(a_i, m_j, \lambda_z)$ values, because communication times in the particular target HiPer-D system are of the same order as computation times.

The next major step in setting up the simulations was to derive reasonable values for the output rates for the sensors and the end-to-end latency constraints for different paths in the system. To that end, both the output rates and end-to-end latency constraints should be related to the values of $s^{\mathrm{c}}(a_i, m_j, \lambda_z)$, $i^{\mathrm{c}}(a_i, m_j, \lambda_z)$, $s^{\mathrm{n}}(a_i, a_j, \lambda_z)$, and $i^{\mathrm{n}}(a_i, a_j, \lambda_z)$ derived above. Moreover, the rates of the HiPer-D system sensors vary widely based on the attack scenario. Therefore, the simulation setup should allow changes in the sensor rates. The following discussion explains the process used in the simulation setup to do this.

First, the rates of all sensors were tentatively set to unity. Then, the DAG showing the connectivity among all applications was scanned to predict the application a_x that is likely to have the most difficulty in meeting its throughput constraint. In particular, let

$$\underline{Z_1(a_i)} = \frac{\sum_{m_j \in \mathcal{M}} \mathrm{ETC}(a_i, m_j, \boldsymbol{\lambda})}{|\mathcal{M}|}$$

$$\underline{Z_2(a_i)} \approx \frac{\sum_{\substack{(m_p, m_q) \text{ pairs} \\ m_p, m_q \in \mathcal{M} \\ m_p \neq m_q}} \max_{a_y \in \mathcal{D}(a_i)} \mathrm{ETK}(a_i, a_y, m_p, m_q, \boldsymbol{\lambda})}{|\mathcal{M}|^2 - |\mathcal{M}|}$$

$$\underline{Z(a_i)} = \max(Z_1(a_i), Z_2(a_i))$$

then $a_x = \mathrm{argmax}_{a_i \in \mathcal{A}} Z(a_i)$.

For a_x to meet its throughput constraint, its computation and communication times must be less than one if all sensor rates are equal to unity. However, if all sensor rates are in fact equal to κ, then the maximum of the computation and communication times of a_x must be less than $1/\kappa$. Conversely, if the maximum of the computation and communication times for a_x is equal to τ, the throughput constraint for a_x will be just met if the rates of all sensors were $1/\tau$. Obviously, the multi-tasked computation and communications times of a_x are not known because machine assignments are not made when the simulations are being setup. Therefore, this study uses $Z(a_x)$ as an estimate for the computation and communication times for a_x. Specifically, the final rates of the sensors were generated by sampling a uniform distribution with the range $[0.75\times\rho,\ \rho]$, where $\underline{\rho} = 1/\left(w_{\mathrm{THR}} \times Z(a_x)\right)$, and $\underline{w_{\mathrm{THR}}} > 0$ is a real number that can be adjusted empirically to "tighten" the throughput constraint. The smaller the value of w_{THR}, the "tighter" the throughput constraint. For a given path $\mathcal{P}_k$, the end-to-end latency constraint, $L_{\max}(\mathcal{P}_k)$, was set so that

$$L_{\max}(\mathcal{P}_k) = w_{\mathrm{LAT}} \times \sum_{a_i \in \mathcal{P}_k} \Big(Z_1(a_i) + Z_2(a_i)\Big),$$

where $\underline{w_{\mathrm{LAT}}} > 0$ is a real number that can be adjusted to change the difficulty of meeting the latency constraint.

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 at least 30 times to obtain good estimates of the mean and standard deviation of $\Delta\Lambda^{\mathrm{SW}}$. Call each repetition of a given experiment a trial. For each new trial, a DAG with $|\mathcal{A}|$ nodes was randomly regenerated, and the values of $s^{\mathrm{c}}(a_i, m_j, \lambda_z)$, $i^{\mathrm{c}}(a_i, m_j, \lambda_z)$, $s^{\mathrm{n}}(a_i, a_j, \lambda_z)$, and $i^{\mathrm{n}}(a_i, a_j, \lambda_z)$ were regenerated from their respective distributions. The maximum fan-in and maximum fan-out values for the DAG generation were two. See [2] for the details of generating the DAG.

The results for a selected set of representative experiments are shown in Figures 7 to 9. Figures 8 and 9 also show the value of failure ratio (FR) for each heuristic, where 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 scale for FR is given on the right-hand side of the graph in these figures. Note that the $\Delta\Lambda^{SW}$ value is averaged only for those trials for which every heuristic successfully found a mapping.

The first two figures compare the heuristics with an exhaustive search (ES) for the optimal mapping. These figures show the average value of $\Delta\Lambda^{SW}$ normalized with respect to the optimal mapping found by ES, along with 95% confidence intervals for a total of 60 trials. The interval shown at the top of each bar is the 95% confidence interval [32]. The normalized $\Delta\Lambda^{SW}$ for a given heuristic is equal to $\Delta\Lambda^{SW}$ for the mapping found by that heuristic divided by $\Delta\Lambda^{SW}$ for the optimal mapping.

Figure 7 shows that, for the given throughput-constrained system, MCTF, TB TPG-X, and TB TPG give very good performance, with the average $\Delta\Lambda^{SW}$ values of 96%, 95%, and 95%, respectively, of the $\Delta\Lambda^{SW}$ value for the optimal mapping. MCPF performs at 55% of the optimal. For this experiment, the Duplex performance was the same as that for MCTF. Note that TB TPG performs better than TPG by a factor of almost three. Recall the two features that distinguish TB TPG from TPG: (a) the special first iteration and (b) the use of application criticality to resolve any ties between applications in the subsequent iterations. Due to the nature of the optimization criterion, ties between applications are common in Line 4 of Figure 5. Because these ties are resolved arbitrarily in TPG, the heuristic makes many arbitrary decisions during its course of execution. Also note in Figure 7 that TPG-X performs more than twice as well as TPG. A probable reason is that like MCTF and TB TPG, TPG-X also first maps the application with the smallest $\Delta\Lambda^{SW*}$ value on its best machine. The FGH-L heuristic, which tries to optimize the maximum allowable load subject to the latency constraint, understandably performs the worst in this throughput-constrained system.

Figure 8 shows the normalized $\Delta\Lambda^{SW}$ for the given latency constrained system. It can be seen that, compared to Figure 7, the MCTF performance falls and the MCPF performance improves. Although MCPF individually approaches only 70% of the optimal mapping, the Duplex heuristic gives an average performance of 80%. The FGH-T heuristic, which tries to optimize the maximum allowable load subject to the throughput constraint, understandably performs the worst in this latency-constrained system.

The comparison of the heuristics for a larger problem is given in Figure 9. The values of w_{LAT} and w_{THR} were adjusted empirically to give a tightly constrained system. In this experiment, FGH-T and FGH-L failed for 82% and 22% of the trials respectively. Given a total of 80 trials, the number of trials in which all heuristic succeeded was too low (ten) to give accurate averages. Therefore, both fast greedy heuristics were excluded from Figure 9 which is plotted for the 52 trials in which all heuristics except the fast greedy ones succeeded. Duplex is the only heuristic that does not fail for any of the 80 trials (FR = 0). It also gives the best value for the average $\Delta\Lambda^{SW}$.

Additional experiments were performed for various other combinations of $|\mathcal{A}|$, $|\mathcal{M}|$, w_{LAT}, and w_{THR}, and the relative performance of Duplex was found to be similar to that shown in Figure 9. In addition, the positive effects of TB TPG were seen more clearly for higher values of the ratio $|\mathcal{A}|/|\mathcal{M}|$, i.e., when a larger number of ties between applications are likely.

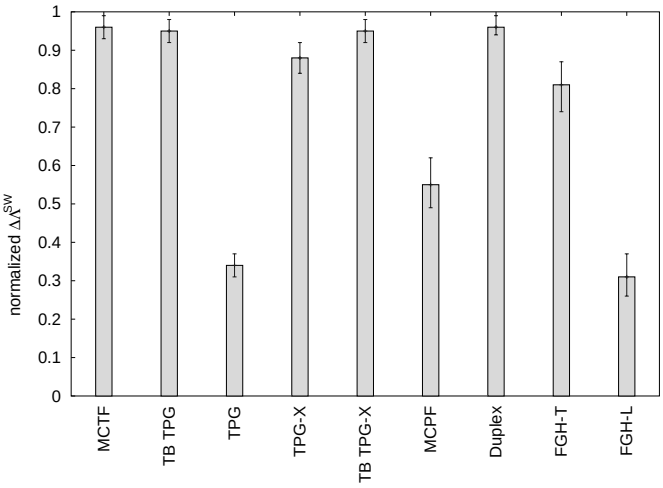

Fig. 7. The average value of normalized $\Delta\Lambda^{SW}$ in a throughput-constrained system. $|\mathcal{M}| = 4$, $|\mathcal{A}| = 14$, $w_{LAT} = 10000$, and $w_{THR} = 10$. Number of sensors = number of actuators = 2.

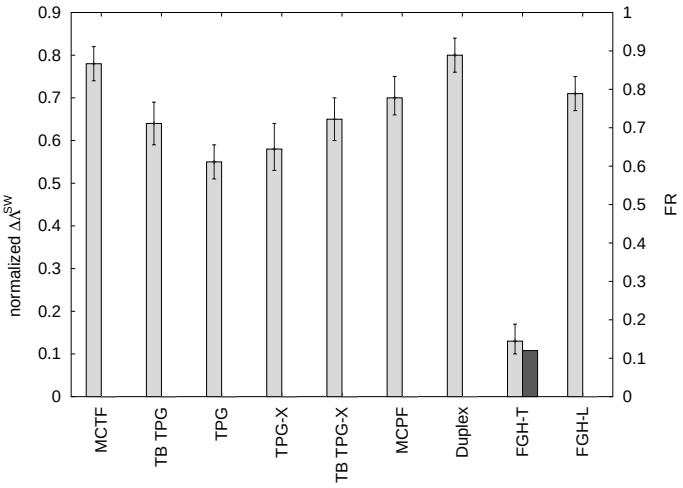

Fig. 8. The average value of normalized $\Delta\Lambda^{SW}$ in a latency-constrained system. All parameters are the same as in Figure 7 except that $w_{LAT} = 10$ and $w_{THR} = 10000$. The second (darker) bar for FGH-T shows FR.

VIII. Conclusions

This research investigates the problem of allocating real-time applications to a set of COTS heterogeneous machines connected together by a COTS high-speed network. For the intended distributed real-time system, the work presented in this paper includes characterizing and modeling the applications and the hardware platform, identifying and quantifying the performance goal, and designing and developing heuristics for allocating the applications so as to attempt to optimize the performance goal.

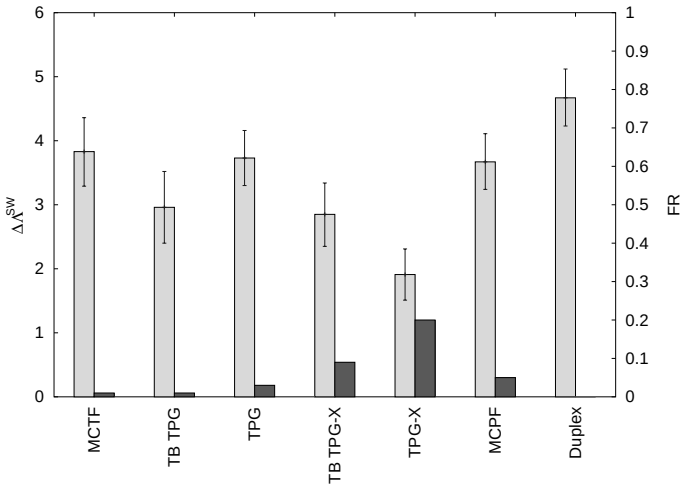

Fig. 9. The average values of $\Delta\Lambda^{\mathrm{SW}}$ and FR in a system where $|\mathcal{M}| = 6$, $|\mathcal{A}| = 50$, $w_{\mathrm{LAT}} = 10$, and $w_{\mathrm{THR}} = 20$. Number of sensors = number of actuators = 4. The second (darker) bar, if present, shows FR.

Each application has certain quality of service (QoS) constraints that must not be violated (i.e., constraints on the end-to-end latency and throughput). Unlike most of the related work in real-time systems, the focus of this work is on finding an initial static allocation of the applications onto the machines to maximize the allowable increase in workload until the dynamic reallocation of resources is required to avoid a QoS violation due to variations in the run-time input-data-dependent parameters. The research uses a performance metric that is suitable for evaluating a statically derived mapping for systems that operate in changing environments. The performance metric quantifies how "well-prepared" the mapping is for absorbing run-time increases in system workload (other run-time parameters may also be used).

This paper also presented and compared three greedy heuristics to solve the initial mapping problem to achieve this performance goal in a HiPer-D-like system. The heuristics were compared under a variety of simulated heterogeneous computing environments. The results from the simulation experiments show that, for small problems, Duplex performs close to the optimal solution. The heuristics were also compared for larger problems sizes (for which exhaustive search could not be performed). For these problems, the Duplex heuristic performed the best. For all of the cases considered, Duplex gave a failure ratio of zero, and a significantly better average value of $\Delta\Lambda^{\mathrm{SW}}$ compared to all other heuristics. The Duplex heuristic has performed the best for the different HC environments discussed here, and is the heuristic of choice for these HC environments, especially when it is critical to find a feasible mapping in highly constrained systems where other heuristics discussed in this paper would fail.

References

[1] S. Ali, J.-K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna. 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 CD-ROM "Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)," Apr. 2002.

[2] S. Ali, J.-K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna. Greedy optimization techniques for resource allocation in dynamic distributed real-time heterogeneous computing systems. Technical report, Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, 2002, in preparation.

[3] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang Journal of Science and Engineering,* Special 50th Anniversary Issue, 3(3):195–207, invited, Nov. 2000.

[4] 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)*, pages 79–87, Mar. 1998.

[5] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Real Time Programming,* W. A. Halang and K. Ramamritham, eds., pages 127–132. Pergamon, Oxford, UK (Proceedings of the IFAC/IFIP Workshop, May 1991), 1992.

[6] I. Banicescu and V. Velusamy. Performance of scheduling scientific applications with adaptive weighted factoring. In *10th IEEE Heterogeneous Computing Workshop (HCW 2001)*, in the CD-ROM "Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)," paper HCW_06, Apr. 2001.

[7] 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 CD-ROM "Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)," paper HCW_15, Apr. 2001.

[8] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *11th Real-Time Systems Symposium*, pages 182–190, Dec. 1990.

[9] R. Bettati and J. W.-S. Liu. End-to-end scheduling to meet deadlines in distributed systems. In *International Conference on Distributed Computing Systems*, pages 452–459, June 1992.

[10] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, 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.

[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)*, pages 1–12 (invited keynote paper), June 2001.

[12] B. M. Carlson and L. W. Dowdy. Static processor allocation in a soft real-time multiprocessor environment. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):316–320, Mar. 1994.

[13] C. D. Cavanaugh, L. R. Welch, B. A. Shirazi, E. Huh, and S. Anwar. Quality of service negotiation for distributed, dynamic real-time systems. In *Parallel and Distributed Processing.* J. Rolim *et al.* eds., Lecture Notes in Computer Science, Vol. 1800, pp. 757–765, Springer-Verlag, New York, NY, 2000.

[14] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, Oct. 1989.

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

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

[17] S. Faucou, A.-M. Deplanche, and J.-P. Beauvais. Heuristic techniques for allocating and scheduling communicating periodic tasks in distributed real-time systems. In *IEEE International Workshop on Factory Communication Systems*, pages 257–265, Sep. 2000.

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

[19] I. Foster. *Designing and Building Parallel Programs.* Addison-Wesley, Reading, MA, 1995.

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

[21] 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)*, pages 184–199, Mar. 1998.

[22] R. F. Freund and H. J. Siegel. Heterogeneous processing. *IEEE Computer*, 26(6):13–17, June 1993.

[23] 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*, in the CD-ROM "Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)," Apr. 2002.

[24] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Fixed priority scheduling periodic tasks with varying execution priority. In *12th Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[25] 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)*, pages 184–198, Apr. 1999.

[26] C.-J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers*, 46(12):1338–1356, Dec. 1997.

[27] R. R. Howell and M. K. Venkatrao. On non-preemptive scheduling of recurring tasks using inserted idle times. *Information and Computation*, 117(1):50–62, Feb. 1995.

[28] C.-W. Hsueh and K.-J. Lin. Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model. *IEEE Transactions on Computers*, 50(1):51–66, Jan. 2001.

[29] E. Huh, L. R. Welch, B. A. Shirazi, B. Tjaden, and C. D. Cavanaugh. Accomodating QoS prediction in an adaptive resource management framework. In *Parallel and Distributed Processing.* J. Rolim *et al.* eds., Lecture Notes in Computer Science, Vol. 1800, pp. 792–799, Springer-Verlag, New York, NY, 2000.

[30] 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.

[31] K. M. J. Islam, B. A. Shirazi, L. R. Welch, B. C. Tjaden, C. Cavanaugh, and S. Anwar. Network load monitoring in distributed systems. In *Parallel and Distributed Processing.* J. Rolim *et al.* eds., Lecture Notes in Computer Science, Vol. 1800, pp. 800–807, Springer-Verlag, New York, NY, 2000.

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

[33] K. Jeffay, D. F. Stanat, and C. U. Martel. On optimal, non-preemptive scheduling of periodic and sporadic tasks. In *12th Real-Time Systems Symposium*, pages 129–139, Dec. 1991.

[34] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th Real-Time Systems Symposium*, pages 201–209, Dec. 1990.

[35] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *10th Real-Time Systems Symposium*, pages 166–171, Dec. 1989.

[36] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *13th Real-Time Systems Symposium*, pages 110–123, Dec. 1992.

[37] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *8th Real-Time Systems Symposium*, pages 261–270, Dec. 1987.

[38] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115–118, Nov. 1980.

[39] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[40] J. W. S. Liu. *Real-Time Systems.* Prentice Hall, Upper Saddle River, NJ, 2000.

[41] 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.

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

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

[44] M. D. Natale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Transactions on Computers*, 49(4):303–316, Apr. 2000.

[45] Naval Surface Warfare Center. In *Report of the HiPer-D C&CA Working Group,* `http://www.nswc.navy.mil/hiperd/reports/thrust/`, 1999.

[46] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling of communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, Dec. 1997.

[47] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–194, Apr. 1990.

[48] M. Saksena and S. Hong. Resource conscious design of distributed real-time systems: An end-to-end approach. In *IEEE International Conference on Engineering of Complex Computer Systems*, pages 306–313, Oct. 1996.

[49] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, Mar. 1994.

[50] L. R. Welch, B. Ravindran, B. A. Shirazi, and C. Bruggeman. Specification and modeling of dynamic, distributed real-time systems. In *19th IEEE Real-Time Systems Symposium (RTSS '98)*, pages 72–81, Dec. 1998.

[51] L. R. Welch and B. A. Shirazi. A dynamic real-time benchmark for assessment of QoS and resource management technology. In *5th IEEE Real-Time Technology and Applications Symposium (RTAS '99)*, pages 36–45, June 1999.

[52] L. R. Welch, B. A. Shirazi, B. Ravindran, and C. Bruggeman. DeSiDeRaTa: QoS management technology for dynamic, scalable, dependable, real-time systems. In *Distributed Computer Control Systems 1998,* F. De Paoli and I. M. MacLeod, eds., pages 7–12. Elsevier Science, Kidlington, UK (Proceedings of the 15th International Federation of Automatic Control (IFAC) Workshop, Sept. 1998), 1999.

[53] L. R. Welch, P. V. Werme, B. Ravindran, L. A. Fontenot, M. W. Masters, D. W. Mills, and B. A. Shirazi. Adaptive QoS and resource management using *a-posteriori* workload characterizations. In *5th IEEE Real-Time Technology and Applications Symposium (RTAS '99)*, pages 266–275, June 1999.

[54] 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.

[55] J. Xu and D. L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, Mar. 1990.

[56] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, C-36(8):949–960, Aug. 1987.