Resource Allocation for Periodic Applications in a Shipboard Environment

Vladimir Shestak[†], Edwin K. P. Chong[†], Anthony A. Maciejewski[†], H. J. Siegel[†§], Lotfi Benmohamed[‡], I-Jeng Wang[‡], and Rose Daley[‡]

Colorado State University
[†]Department of Electrical & Computer Engineering
[§]Department of Computer Science
Fort Collins, CO 80523-1373
{shestak, echong, aam, hj}@engr.colostate.edu

The Johns Hopkins University
[‡]Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099
{lotfi.benmohamed, i-jeng.wang, rose.daley}@jhuapl.edu

Abstract

Providing efficient workload management is an important issue for a large-scale heterogeneous distributed computing environment where a set of periodic applications is executed. The considered distributed system is expected to operate in an environment where the input workload is likely to change unpredictably, possibly invalidating a resource allocation that was based on the initial workload estimate. The tasks consist of multiple application strings, each made up of an ordered sequence of applications. There are quality of service (QoS) constraints that must be satisfied for each string. This work addresses the problem of finding a robust initial allocation of resources to application strings that is able to absorb some level of unknown input workload increase without rescheduling. An allocation feasibility analysis is presented followed by four heuristics for finding a near-optimal allocation of resources. The performance of the proposed heuristics is evaluated and compared using simulation. The proposed heuristics also are compared to a mathematically derived upper bound.

1. Introduction and Problem Statement

The Adaptive and Reflective Middleware Systems (ARMS) program supported by DARPA includes designing an efficient resource allocation capability for a Total Ship Computing Environment (TSCE). One objective of this research program, which is a collaborative effort of many technology developers, is to investigate the problem of robust resource allocation in this class of heterogeneous computing systems. In this paper, we consider a subset of the actual TSCE and application model being examined in ARMS.

The TSCE consists of a set of heterogeneous machines, heterogeneous network links, continuously running periodic applications, and a number of quality of service (QoS) constraints that must be satisfied during the operation of the system. Generally, the TSCE system operates in an environment that undergoes unpredictable changes, e.g., in the system input workload, which may cause QoS violations. Therefore, even though a good initial allocation of resources to applications may ensure that no QoS constraints are violated when the system is first put into operation, dynamic mapping approaches may be needed to reallocate resources during execution (e.g., [22, 26]).

Two different resource allocation schemes were considered in this work. Partial resource allocation occurs when some part of a given set of applications considered for mapping cannot be allocated due to limited system resources or QoS constraint violations. In contrast, complete resource allocation takes place in a system that has enough resources to accommodate all applications considered for mapping without violation of any of the imposed QoS constraints. The goal of this work is to find the "best" static initial mapping (i.e., one found during an off-line planning phase) of applications to computing and networking resources, which maximizes the "total worth" of the system's performance and the system's capacity to absorb

This research was supported by the DARPA Information Exploitation Office under contract No. NBCHC030237, by the Colorado State University George T. Abell Endowment, and by the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology). Approved for public release, distribution unlimited.

unpredictable increases in input workload without QoS violations (explained in Section 4).

The development of heuristic techniques to find near-optimal solutions for resource allocation problem is an active area of research (e.g., [1, 6, 7, 17, 25, 28, 30]). For the intended distributed system, the contributions of this work include developing a method to analyze the feasibility of an allocation, quantifying the performance goal, designing and developing heuristics for mapping the applications to resources to optimize the performance goal, evaluating the relative performance of these heuristics, and deriving mathematical bounds in performance.

The reminder of this paper is organized in the following manner. Section 2 develops models for the applications and hardware platform. Sections 3 and 4 present a quantitative basis for the feasibility analysis and performance measure for a given resource allocation, respectively. Four heuristics to solve the posed initial mapping problem are described in Section 5, followed by the simulation setup in Section 6. A mathematical model for finding performance upper bounds in different simulation scenarios is provided in Section 7. The simulation experiments and performance evaluation of the heuristics are discussed in Section 8. A sampling of some related work is presented in Section 9. Section 10 concludes the paper.

2. System Model

The overall TSCE system is composed of a number of heterogeneous computational resources distributed across a shipboard environment and connected by a communication network [1]. Details behind the real communication network are outside the scope of this paper, where its functionality is modeled via all possible independent virtual point-to-point communication routes, each characterized by a maximum available bandwidth. The existing networking technologies can enforce that communication model through resource reservations at system initialization time. Each machine in the system is capable of multitasking. Similarly, a given communication route is shared among multiple active data transmissions traversing that communication route.

In the TSCE system, a string is defined as a continuously executing sequence of applications connected in precedence order by specified data transfers [2]. Data is received by a string from other strings or from sensors with a fixed period. The output produced by the string serves as an input to other application strings or to actuators.

Let S^k be the k^{th} string, specified by a sequence of n_k applications: $S^k = a_1^k a_2^k \ldots a_{n_k}^k$. To model the importance each string represents in the system, for each k, the k^{th} string is preassigned one of three possible worth factors, $I[k] \in \{1, 10, 100\}$. Let $P[k]$ be the period associated with string S^k, where each a_i^k must execute once each period. The minimum throughput constraint states that the computation time of any application or the time of any inter-application data transfer in S^k is required to be no larger than $P[k]$. For each string S^k the maximum end-to-end latency constraint $L^{\max}[k]$ is specified as a limit on the total amount of time for a given data set to be sequentially processed by string S^k. Assuming that the resource allocation for string S^k is made, let $m[i,k]$ denote the machine to which application a_i^k is assigned. Let $t_{comp}^k[i]$ be the estimated computation time for application a_i^k for each data set (executing on $m[i,k]$). Let $t_{tran}^k[i]$ be the estimated transfer time required to send output of size $O^k[i]$ from application a_i^k (on $m[i,k]$) to application a_{i+1}^k (on $m[i+1,k]$) within string S^k. A typical allocation of string S^k in the system is illustrated in Figure 1 below.

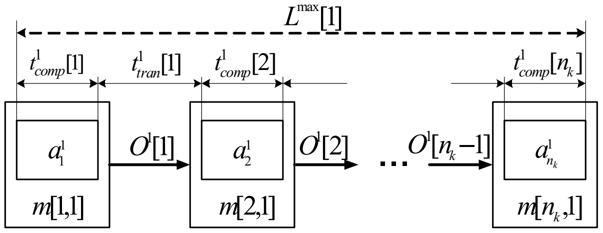

Figure 1. The string model. Rectangles denote applications in the string and the machines where they execute. The arrows represent output data transfers within the string.

Mathematically, for a given resource allocation for a string, the aforementioned constraints are satisfied if:

[1] In the final ARMS system, computational resources will be divided into pools; in this paper, we assume each pool consists of one machine.

[2] The final ARMS program may include DAGs of applications.

$$\begin{cases} t^k_{comp}[i] \le P[k] & 1 \le i \le n_k \\ t^k_{tran}[i] \le P[k] & 1 \le i \le n_k - 1 \\ t^k_{comp}[n_k] + \sum_{i=1}^{n_k-1}\left(t^k_{comp}[i] + t^k_{tran}[i]\right) \le L^{\max}[k] \end{cases} \quad (1)$$

If these constraints are satisfied for an allocation, the allocation is said to be feasible. Because both machines and communication routes are assumed to be shared, $t^k_{comp}[i]$ and $t^k_{tran}[i]$ will depend on the level of sharing—i.e., the number of applications assigned to a computational resource and currently active, or the number of current data transfers assigned to a communication route. Furthermore, these values will depend on how an application or a data transfer is prioritized by a machine's or network's local scheduler with respect to all other applications or data transfers that share this computation or communication resource.

3. Allocation Feasibility Analysis

To validate the feasibility of resource allocations produced by the proposed heuristics, the following two-stage feasibility analysis was developed. The first-stage analysis verifies whether the overall utilization of both computation and communication resources is below the full system's capacity.

Two parameters are used in the TSCE environment to specify the workload imposed by each application on a particular machine: the nominal execution time and the nominal CPU utilization. The nominal execution time $\underline{\bar{t}^k[i,j]}$ is the time required by application a^k_i in string S^k to process a data set when a^k_i is the only application executing on machine j. Hence, $t^k_{comp}[i] \ge \bar{t}^k[i, m[i,k]]$. The nominal CPU utilization $\underline{\bar{u}^k[i,j]}$ is the average CPU utilization of machine j during that execution. The product $\bar{t}^k[i,j] \times \bar{u}[i,j]$ can be interpreted as the fixed amount of CPU work required for application a^k_i to process a data set on machine j. This fixed amount of CPU work can be performed in many different ways. For example, if only half of $\bar{u}^k[i,j]$ is allocated, then the execution time required to accomplish the same fixed amount of CPU work is twice $\bar{t}^k[i,j]$.

Let the conditional **1** function be defined by:

$$\mathbf{1}(condition) \equiv \begin{cases} 1 & \text{if } condition \text{ is true;} \\ 0 & \text{otherwise.} \end{cases}$$

If $\underline{A}$ strings are allocated in the system then the overall machine utilization $\underline{U^{machine}[j]}$ is computed as:

$$U^{machine}[j] = \sum_{k=1}^{A}\sum_{i=1}^{n_k}\left(\frac{\bar{t}^k[i,j]}{P[k]} \times \bar{u}^k[i,j] \times \mathbf{1}(m[i,k] = j)\right) \quad (2)$$

The term $\frac{\bar{t}^k[i,j]}{P[k]} \times \bar{u}^k[i,j]$ provides the average CPU utilization allocated for application a^k_i over $P[k]$. This is the minimum average CPU utilization that allows a^k_i to complete processing without a throughput constraint violation. The sum of such minimum CPU utilizations across all the applications executing on j determines the overall machine utilization.

If $\underline{w[j_1, j_2]}$ denotes the total bandwidth of communication route from machine j_1 to machine j_2, the overall communication route utilization $\underline{U^{route}[j_1, j_2]}$ is:

$$U^{route}[j_1, j_2] = \frac{1}{w[j_1, j_2]} \times \sum_{k=1}^{A}\sum_{i=1}^{n_k-1}\left(\frac{O^k[i]}{P[k]} \times \mathbf{1}(m[i,k] = j_1 \; \& \; m[i+1,k] = j_2)\right) \quad (3)$$

The term $\frac{O^k[i]}{P[k]}$ can be interpreted as the minimum average bandwidth allocated to application a^k_i for output transfer over $P[k]$ that allows it to be completed without a throughput constraint violation.

For a given allocation, the result of the first-stage analysis is considered satisfied if the computed utilization value is no larger than one for each machine and each communication route.

The second-stage analysis focuses on checking the throughput and end-to-end latency constraints for each allocated string and relies on an assumption that scheduling at the machine level is carried out based on the "relative tightness" of a string. Relative tightness $\underline{T[k]}$ is the ratio of the total time required for a data set to be processed by string S^k (no sharing assumed on computation and communication resources) to the end-to-end latency constraint $L^{\max}[k]$ specified for this application string, determined as:

$$T[k] = \frac{1}{L^{\max}[k]} \times \left[\bar{t}^{k}[n_k, m[n_k,k]] + \sum_{i=1}^{n_k - 1} \left(\bar{t}^{k}[i, m[i,k]] + \frac{O^k[i]}{w[m[i,k], m[i+1,k]]} \right) \right] \quad (4)$$

Assume that the local scheduling policy is based on the rule that applications and data transfers with higher relative tightness values are given higher execution priorities and will be scheduled first. This analysis can be modified if a different scheduling policy is used. Without loss of generality, assume that the $T[k]$ values are distinct.

To provide insight into the method of computing $t^k_{comp}[i]$, consider the example, depicted in Figure 2, with three possible cases of CPU sharing on machine j. Two applications a_1^1 and a_1^2, illustrated in this example, belong to different strings (1 and 2) and share CPU resources of machine j. To capture the worst-case overlap between processes, periods $P[1]$ and $P[2]$ are lined up at their beginnings, and each application receives an input at the beginning of its period. Suppose that string 1 (to which application a_1^1 belongs) is relatively tighter than string 2. Based on the local scheduling policy, application a_1^1 has a higher execution priority for the CPU than application a_1^2. As a result, a_1^1's estimated computation time is not affected by CPU sharing—i.e., $t^1_{comp}[1] = \bar{t}^1[1,j]$.

The simplest case of overlap is illustrated in case (1) in Figure 2. Strings 1 and 2 have the same periods, and a_1^1 and a_1^1 are able to utilize 100% of the CPU, i.e., $\bar{u}^1[1,j] = \bar{u}^2[1,j] = 1$. Taking into account the delay caused by the execution of application a_1^1, the estimated computation time for application a_1^2 can be found as $t^2_{comp}[1] = \bar{t}^2[1,j] + \bar{t}^1[1,j]$. Thus, a_1^2 must wait for a time interval of $t^1_{comp}[1]$ before it can begin executing for each period.

Now suppose that applications a_1^1 and a_1^2 have different periods, while all other parameters remain the same. For example as shown in case (2), $P[1]$ is twice $P[2]$. Note that the processing of only every other data set of application a_1^2 is delayed now. Therefore, the average waiting time for application a_1^2 in this case becomes affected by the ratio of $P[2]$ to $P[1]$, and can be expressed as $\frac{P[2]}{P[1]} \times \bar{t}^1[1,j]$.

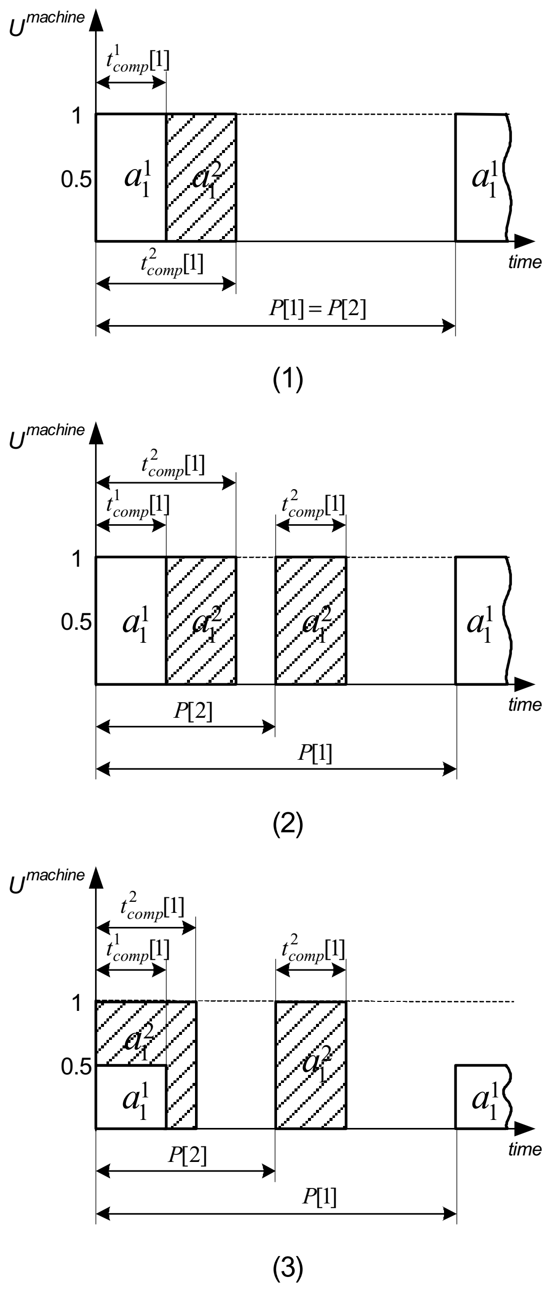

Figure 2. Three possible cases of data set processing overlaps resultant from CPU sharing between applications a_1^1 and a_1^2. Application a_1^1 has a higher execution priority than application a_1^2.

In case (3), applications execute under the same conditions as in case (2), except that application a_1^1 is able to utilize at most 50% of the CPU on average, i.e., $\bar{u}^1[1,j] = 0.5$. This fact allows part of application a_1^2 to execute concurrently with a_1^1 by utilizing the remaining 50% of the CPU cycles. Application a_1^2 completes processing each overlapping data set earlier that it would have if a_1^1 was to require 100% of the CPU. This observation allows the average waiting time (hypothetical in this case) to be determined by including the dependence on $\bar{u}^1[1,j]$ into the expression derived in case 2. Specifically, the average waiting time is $\bar{u}^1[1,j] \times \dfrac{P[2]}{P[1]} \times \bar{t}^1[1,j]$.

Based on the examples discussed above, equation (5) below is one way to estimate computation time for each prioritized periodic application executing on a shared CPU:

$$t_{comp}^k[i] = \bar{t}^k[i,m[i,k]] + \sum_{z=1}^{A} \frac{P[k]}{P[z]} \times \sum_{p=1}^{n_z} \left(\begin{array}{l} \bar{t}^z[p,m[p,z]] \times \bar{u}^z[p,m[p,z]] \times \\ \mathbf{1}(m[p,z] = m[i,k] \;\; \& \;\; T[z] > T[k]) \end{array} \right) \tag{5}$$

Similarly, the estimated transfer time for each prioritized periodic output transfer traversing over a shared communication route is:

$$t_{tran}^k[i] = \frac{O^k[i]}{w[m[i,k],m[i+1,k]]} + \sum_{z=1}^{A} \frac{P[k]}{P[z]} \times \sum_{p=1}^{n_z - 1} \left(\begin{array}{l} \dfrac{O^z[p]}{w[m[p,z],m[p+1,z]]} \times \\ \mathbf{1}(m[i,k] = m[p,z] \;\; \& \;\; m[i+1,k] = m[p+1,z]] \;\; \& \\ T[z] > T[k]) \end{array} \right) \tag{6}$$

The first term in equations (5) and (6) represents the nominal time required for an application to process or transfer a data set, respectively. The second term quantifies the average waiting time before the processor or communication transmitter becomes available for a data set. The accuracy of the time estimates produced by these equations depends on many factors related to a particular environment. For example, in equation (5), it depends on how the data arrivals of different applications are relatively phased and to what extent each application deviates from the corresponding nominal CPU utilization value at each point in time. Furthermore, a variety of different local schedulers are possible within each processor. If specific schedulers were known, equations (5) and (6) could be adjusted accordingly.

For the second-stage analysis, a given allocation is considered feasible if for each string S^k the estimated computation and data transfer times satisfy the constraints in (1).

4. Performance Goal

In the context of the intended system, the performance metric for evaluating an application-to-machine mapping generated by the heuristics has two components. The primary component is total worth, defined as the sum of the worth factors associated with strings in the mapping that successfully passed the two-stage feasibility analysis. The secondary component is system slackness. Let Ω be the set composed of all computation and communication resources in the system. System slackness Λ is a measure of the minimum utilization capacity remaining across all of the hardware resources in the set Ω. It quantitatively reflects the system's potential to absorb unpredictable increases in input workload and is defined as:

$$\Lambda = \min \left(\begin{array}{l} \{1 - U^{machine}[j] \; : \; j \in \Omega\} \\ \bigcup \quad \{1 - U^{route}[j_1, j_2] \; : \; j_1, j_2 \in \Omega\} \end{array} \right) \tag{7}$$

The goal of each of the heuristics in section 5 is to achieve the highest level for the primary component while maximizing system slackness Λ at that level.

With the given "worth" scheme, a high worth string has the same value as 10 medium worth strings. A different, alternate scheme is possible, where higher worth strings have a value of more than the total value of any number of strings of medium or low worth. In such a scheme, high worth strings can be put in a special class. The content of this class is allocated first in the system. Such a scheme, described in [25], is outside the current requirements of this work.

5. Descriptions of Heuristics

This section develops four heuristics for the problem of finding an initial static allocation, and begins by introducing some additional terms. Suppose that a set of strings considered for mapping is given. Let the permutation space be all possible orders of the given strings, and let the solution space be all possible application-to-machine assignments. An allocation is considered feasible if the two-stage feasibility analysis presented in Section 3 is satisfied for this allocation. It was observed experimentally a genetic algorithm [30], operating in the solution space, failed to find any feasible allocation even for a relatively small set of strings in the reasonable amount of time. Therefore, the first four heuristics presented in this section search over the permutation space instead of directly over the solution space. A permutation ordering of strings in the permutation space is translated into a mapping in the solution space by applying the Incremental Mapping Routine (IMR). The IMR maps a single string. The four heuristics differ in the order in which the strings are mapped by the IMR.

Incremental Mapping Routine: The allocation algorithm used in the IMR heuristic is based on the greedy mapping technique. The IMR handles one string at a time, retrieving applications in the string for mapping in a certain order, and having its resource-candidate search guided by impact on the resource utilization. Starting from the most computationally intensive application, determined by the equation in step 1 in the pseudo code shown below, the heuristic maps all the intermediate applications along the string up to the next most computationally intensive application. In selecting a mapping, a parameter of interest is the maximum value of the resource utilizations (given by equations (2) and (3)) in the machine-route pair affected by an application assignment. The selection process determines a machine for mapping by finding the minimum value of this parameter across all machines in the system, with ties broken arbitrarily. Then, the next unassigned most computationally intensive application is found, and the same mapping procedure is repeated until the allocation for a given string is completed. The IMR approach attempts to map computationally intensive applications early, but also maps their neighboring applications, so that network utilization also can be taken into account as the heuristic progresses.

To describe the IMR heuristic in detail some additional notation must be introduced. Let $U^{machine}[j,i,k]$ be the utilization of machine j if application a_i^k was assigned to machine j (in addition to the applications assigned previously to this machine). Similarly, let $U^{route}[j_1,j_2,i,k]$ be the utilization of communication route if application a_i^k was assigned to machine j_1 and passed an output to its successor mapped on machine j_2. Assuming M machines in the system, the average nominal execution time $\overline{t}_{av}^k[i]$ and average nominal machine CPU utilization requirement for application a_i^k $\overline{u}_{av}^k[i]$ are defined as follows:

$$\overline{t}_{av}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \overline{t}^k[i,j] \qquad (8)$$

$$\overline{u}_{av}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \overline{u}^k[i,j] \qquad (9)$$

The IMR heuristic can be summarized by the following procedure.

```
1. As a starting point, identify application $a_{i_{\max}}^k$ in the
   given string $S^k$ as follows:
   $i_{\max} = \arg\max\{\frac{\overline{t}_{av}^k[i] \times \overline{u}_{av}^k[i]}{P[k]} : i = 1, ..., n_k\}$.
2. Assign application $a_{i_{\max}}^k$ to the machine $m[i_{\max},k]$
   found as:
   $m[i_{\max},k] = \arg\min\{U^{machine}[j,i_{\max},k] :$
   $j = 1, ..., M\}$.
3. Initialize set $D = \{a_{i_{\max}}^k\}$.
4. **While** set $D$ does not contain all applications in the
   given string $S^k$ **do**
   a. $i_{right}$ = max application index in $D$;
      $i_{left}$ = min application index in $D$;
   b. identify unassigned application $a_{i_{\max}}^k$ in the
      given string $S^k$ as follows:
      $i_{\max} = \arg\max\{\frac{\overline{u}_{av}^k[i] \times \overline{t}_{av}^k[i]}{P[k]} :$
      $i = 1, ..., n_k \ \& \ a_i^k \notin D\}$;
   c. **while** $i_{\max} > i_{right}$ **do**
      ▪ $i_{right} = i_{right} + 1$
      ▪ assign $a_{i_{right}}^k$ to the machine $m[i_{right},k]$
        found as follows:
        $m[i_{right},k] =$
        $\arg\min \max\{U^{machine}[j,i_{right},k],$
        $U^{route}[m[i_{right}-1,k],j,i_{right},k] : 1 \le j \le M\}$;
      ▪ include application $a_{i_{right}}^k$ in set $D$;
```

```
d. while $i_{\max} < i_{left}$ do
    ▪ $i_{left} = i_{left} - 1$
    ▪ assign $a^k_{i_{left}}$ to the machine $m[i_{left}, k]$
      found as follows:
      $m[i_{left}, k] =$
      $\arg\min \max\{U^{machine}[j, i_{left}, k],$
      $U^{route}[j, m[i_{left}+1, k], i_{left}, k] \ : \ 1 \le j \le M\};$
    ▪ include application $a^k_{i_{left}}$ in set $D$.
```

***Most Worth First Heuristic*:** The MWF heuristic begins by ranking strings in the order of their worth. Then, it uses the IMR heuristic to map applications within the current string and applies the two-stage feasibility analysis to each intermediate mapping produced. If a given intermediate mapping fails to pass the feasibility test, the mapping process is terminated, and the previous intermediate mapping is considered as a final result. Thus, in such a case, only a subset of the strings will be mapped.

***Tightest First Heuristic*:** The TF heuristic is conceptually similar to MWF, but differs in the criterion used for ranking. The chosen ranking criterion here is relative tightness, given by equation (4). It is important to note that the procedure given in Section 3 for calculating relative tightness assumes that the complete allocation of all strings in the system is known. Thus, to be used as a ranking criterion, equation (4) is modified such that all terms related to a specific allocation are replaced with average values. The average inverse of bandwidth is determined as an average across all possible communication routes in the system:

$$\frac{1}{w_{av}} = \frac{1}{M^2} \times \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \frac{1}{w[j_1, j_2]}$$

Equations (8) and (9) provide the average nominal execution time and average nominal machine CPU utilization requirement for application a_i^k.

***Permutation Space GENITOR-Based Heuristic*:** This PSG heuristic was developed by combining the IMR heuristic with concepts from the GENITOR approach. GENITOR is an evolutionary steady-state genetic search algorithm that has been shown to work well for several problem domains (e.g., [6, 24, 32]). For the TSCE problem, each chromosome in the heuristic represents an ordered list of strings in the permutation space. GENITOR-specific operators, such as selection, crossover, and mutation, are applied in that space. Chromosomes differ in their list orders, which results in different mappings in the solution space obtained via "projecting" a chromosome to the solution space by applying the IMR. The two-component performance metric defined in Section 4 is used to measure the fitness of each chromosome.

The PSG heuristic was implemented as follows. First, an initial population composed of 250 different chromosomes is generated randomly by reordering the initial given set of strings. After evaluation, the entire set of chromosomes is sorted (ranked) by their fitness. Next, a special function (described later) is used to select two chromosomes to act as parents. These two parents perform a crossover operation, and two offspring are generated. The offspring are then evaluated and must immediately compete for inclusion in the population. If the considered (single) offspring has a higher fitness than the poorest member in the population, the offspring is inserted in sorted order in the population, and the poorest chromosome is removed. Otherwise, the offspring is discarded.

In the crossover step, for the pair of the selected parent chromosomes a random cut-off point is generated that divides the chromosomes into top and bottom parts. Then, the strings in each top part are reordered. The new ordering of the strings in one top part is the relative positions of these strings in the other parent chromosome in the pair. It is important to note the choice of the top parts of the parent chromosomes for reordering. This allows the offspring to differ from their parents in the case of a partial resource allocation. In such a case some strings in the bottom part of each chromosome cannot be mapped, and, as a result, the reordering among these strings will not be reflected in the solution space.

After each crossover, the same special function (described below) is applied again to select a chromosome for mutation. A mutation operator generates a single offspring by perturbing the original chromosome order via swapping two randomly chosen application strings. The resultant offspring is considered for inclusion in the population in the same fashion as for an offspring generated by crossover.

The special function for selecting parent chromosome(s) is a bias function, used to provide a specific selective pressure [32]. For example a bias of 1.5 implies that the top ranked chromosome in the population is 1.5 times more likely to be selected for a crossover or mutation than the median chromosome. The bias value 1.6 was found experimentally by observing the performance of the heuristic while varying the bias values across the range [1,2] in steps 0.1. Elitism, the property of guaranteeing the best solution remains in the population [29], is implicitly implemented by always removing the poorest chromosome.

As the PSG runs, the crossover operator will be sequentially repeated followed by the mutation operator until one of the stopping conditions is reached. The stopping conditions chosen in this study for the PSG heuristic are: (1) 5,000 iterations (where an iteration is one crossover and one mutation), (2) 300 iterations without a change in the elite (best) chromosome, or (3) all chromosomes converged to the same solution.

***Seeded PSG Heuristic*:** The solutions (seeds) found by the MWF and TF heuristics were included in the initial population of the Seeded PSG. All other operations and stopping conditions remained identical to the PSG described above.

6. Simulation Setup

The purpose of this study was to evaluate the performance of the mapping heuristics in three different workload scenarios. For all these scenarios, the hardware part of the intended system was composed of a heterogeneous suite of 12 machines. The bandwidth of each inter-machine communication route was chosen by sampling a uniform distribution in the interval between 1 and 10 Mb/sec. All intra-machine communication routes were assumed to have infinite bandwidth. In addition, the time-of-flight—i.e., time needed for a transmitted bit of data to reach the destination [20]—was assumed to be negligible on each communication route. For all the experiments, it also was assumed that an application could execute on any machine, and an output could be transferred over any communication route. Each of the three workload scenarios was distinguished by a different number of strings considered for mapping and different specification ranges for each string's period and end-to-end latency constraint.

In the first scenario, the modeled system is considered highly loaded—i.e., not all strings in a given set can be successfully allocated due to the fact that some hardware component in the system reaches its computation or communication capacity limit. As an indicator, the first-stage feasibility analysis becomes unsatisfied at this point, and the sequential string allocation process is stopped. To model this situation, a set consisting of 150 strings was generated. The throughput and latency parameters were intentionally relaxed in this scenario for each string to avoid any possible stoppage of a mapping process caused by the QoS constraints violation.

The second type of partial mapping of a given set of strings is investigated in the second simulation scenario by modeling a QoS-limited system. In this scenario, 150 strings were used again, but the throughput and latency constraints associated with them were set tighter. As such, the allocation process is forced to stop before any of the system's hardware resources reaches its capacity limit.

The third simulation scenario involves a lightly loaded system where the entire set of strings can be allocated—i.e., it contains only 25 strings with relaxed QoS constraints. Clearly, because a complete mapping is achievable now, only the secondary component of the performance metric, i.e., the system slackness, matters in this scenario.

For each scenario, 100 simulation runs were performed, resulting in reasonably tight 95% confidence intervals [14]. In each simulation run, the heuristics were tested using strings composed of a different number of applications determined randomly within the range from 1 to 10. The nominal execution time and nominal machine CPU utilization requirement associated with each application in the string were set by sampling a uniform distribution in the intervals between 1 and 10 seconds, and between 0.1 and 1, respectively. In the same fashion, the size of a data output generated by each application in the string was chosen in the interval from 10 to 100 Kbytes.

7. Upper Bound Calculation

In each scenario, the performance of the proposed allocation heuristics was compared with a mathematically derived upper bound, UB. The upper bound was computed by allowing "fractional mappings." In this method, it is assumed that every application considered for mapping in the suite of M machines can be decomposed into any M fractions. Each of these M fractions corresponds to a particular machine and represents the part of the application assigned to this machine. It also is assumed that an application fraction assigned to a machine receives the equivalent fraction of the application's data input and produces the equivalent fraction of the application's data output. For example, if $2/3$ of application a_i^k is allocated to machine j then this fraction receives $2/3$ of the application's input $O^k[i-1]$ and produces $2/3$ of the application's output $O^k[i]$. These assumptions allow the problem of finding the UB to be solved by applying a Linear Programming (LP) approach, e.g., the Simplex algorithm [12] or one of the interior-points methods [18].

Specifically, the employed LP algorithm operates with fractions of applications and fractions of application outputs, considering them as decision variables and producing the fractional mapping that maximizes the performance metric. Remarkably, the global optimal solution in LP problems can be found in polynomial time [8]. Of course, this fractional allocation cannot be implemented in practice because, in general, applications are not fractionally decomposable in the TSCE environment.

To specify mathematically the LP problem of finding the UB, some additional notation needs to be defined. Let $\underline{x[i,k,j]}$ be the fraction of application a_i^k assigned to machine j. Similarly, let $\underline{y[i,k,j_1,j_2]}$ be the fraction of the output generated by a_i^k assigned to the communication route from machine j_1 to machine j_2. Because application assignments are considered on a fractional level now, equation (2) for machine utilization and equation (3) for communication route utilization need to be restated:

$$U^{machine}[j] = \sum_{k=1}^{A}\sum_{i=1}^{n_k}\left(\begin{array}{l}\dfrac{\bar{t}^k[i,j]}{P[k]} \times \bar{u}^k[i,j] \times \\ x[i,k,j]\end{array}\right) \qquad (10)$$

$$U^{route}[j_1,j_2] = \frac{1}{w[j_1,j_2]} \times \sum_{k=1}^{A}\sum_{i=1}^{n_k-1}\left(\frac{O^k[i]}{P[k]} \times y[i,k,j_1,j_2]\right) \qquad (11)$$

Suppose that $\underline{Q}$ represents the total number of strings considered for mapping in the system. In the case of partial resource allocation (simulation scenarios 1 and 2), the primary objective was to *maximize* the total worth of the strings deployed in the system, as defined in Section 4. This transforms into the formal LP representation of an objective function:

Maximize

$$\sum_{k=1}^{Q}\sum_{i=1}^{n_k}\left(I[k] \times \sum_{j=1}^{M} x[i,k,j]\right)$$

The optimization problem is *subject to* the set of constraints (a)–(g), explained in detail below.

$$\text{For } 1 \le k \le Q, \quad \sum_{j=1}^{M} x[1,k,j] \le 1 \qquad (a)$$

$$\text{For } 2 \le i \le n_k,\ 1 \le k \le Q, \quad \sum_{j=1}^{M} x[i,k,j] = \sum_{j=1}^{M} x[1,k,j] \qquad (b)$$

$$\text{For } 1 \le i \le n_k,\ 1 \le k \le Q,\ 1 \le j \le M, \quad x[i,k,j] \ge 0 \qquad (c)$$

$$\text{For } 1 \le i \le n_k - 1,\ 1 \le k \le Q,\ 1 \le j_1 \le M, \quad x[i,k,j_1] = \sum_{j_2=1}^{M} y[i,k,j_1,j_2] \qquad (d)$$

$$\text{For } 2 \le i \le n_k,\ 1 \le k \le Q,\ 1 \le j_2 \le M, \quad x[i,k,j_2] = \sum_{j_1=1}^{M} y[i,k,j_1,j_2] \qquad (e)$$

$$\text{For } 1 \le j \le M, \quad U^{machine}[j] \le 1 \qquad (f)$$

$$\text{For } 1 \le j_1, j_2 \le M, \quad U^{route}[j_1,j_2] \le 1 \qquad (g)$$

Condition (a) bounds the range of possible values of the x decision variables related to the first application in each string. Note that condition (a) is specified with an inequality, which implies that applications can be mapped *partially*—otherwise, no feasible solution will be found in the case of partial resource allocation. Condition (b) requires the same fraction of all applications in the same string be mapped. Condition (c) takes out of consideration any possible negative solutions. Condition (d) requires that an application fraction assigned to a machine produces the equivalent fraction of the application's data output. Similarly, condition (e) stipulates that an application fraction assigned to a machine receives the equivalent fraction of the application's data input. Condition (b) and (e) apply to the strings that are composed of more than one application. The enforcement of the first-stage feasibility analysis is represented by the remaining two conditions (f) and (g). These conditions bound the total CPU utilization on each machine and the total transfer utilization on each communication route. Note that conditions (f) and (g) are based on equations (10) and (11), respectively.

In the case of complete resource allocation (simulation scenario 3), the objective was to *maximize* system slackness Λ, given by (7). Thus, the objective function in the LP problem is formally stated as:

Maximize Λ

The optimization problem is *subject to* constraints (b)–(g), which remain identical to the case of partial resource allocation considered above. In contrast to that case, condition (a) needs to be restated to force all applications to be completely mapped:

$$\text{For } 1 \le i \le n_k,\ 1 \le k \le Q, \quad \sum_{j=1}^{M} x[i,k,j] = 1$$

Note that the described optimization method for UB computation can be potentially applied to the problem of finding the actual mapping in scenarios (1) and (3) by introducing an additional condition:

$$\text{For } 1 \le i \le n_k,\ 1 \le k \le Q,\ 1 \le j \le M,$$
$$x[i,k,j] \in \{0,1\}$$

However, with such a condition, the problem falls into the category of NP-complete Integer Programming (IP) problems [27]. It is important to note that because the global optimal solution for the LP version of a given problem in each scenario is indeed the upper bound for the more restricted IP one, the proposed method of computing UB is mathematically justified.

8. Experimental Results

An interactive software application has been developed that allows simulation, testing, and demonstration of the heuristics examined in Section 5. The software allows a user to specify the number of machines *M*, the number of strings *Q*, and the maximum possible number of applications in the string. Let μ be a random variable chosen in a particular range. The end-to-end latency constraint $L^{\max}[k]$ for each generated string was assigned in the experiments as follows:

$$L^{\max}[k] = \mu \times \left[t_{av}^{k}[n_k] + \sum_{i=1}^{n_k-1} \left(t_{av}^{k}[i] + \frac{O^k[i]}{w_{av}} \right) \right]$$

The period $P[k]$ associated with each string was set in the experiments as follows:

$$P[k] = \mu \times \max\{ t_{av}^{k}[i], \frac{O^k[z]}{w_{av}} : 1 \le i \le n_k,\ 1 \le z \le n_k - 1 \}$$

The ranges for the random variable μ specified with respect to a simulation scenario executed are shown in Table 1. The commercial optimization package Lingo 9.0 was applied to the generated workload parameters in each simulation run to compute the corresponding upper bounds, as described in Section 7.

The experimental results from multiple simulation runs are illustrated in Figures 3, 4, and 5. In simulation scenarios 1 and 2, the experimental results represent the total worth achieved for the partial mapping averaged across 100 simulation runs. In each simulation run, only the best result of the four different trials produced by the evolutionary algorithms (PSG and Seeded PSG) contributed to the graph statistics. As such, four different starting points in the permutation space were established allowing each algorithm to iterate in four different convergence paths. The performance results of the proposed heuristics in these two simulation scenarios were compared with the upper bounds indicating the maximum performance achievable. The largest difference between the performance of heuristics and computed upper bounds was observed in simulation scenario 2, where a QoS violation occurred before any of the resources reached its maximum computation or communication capacity. Results for simulation scenario 3 represent system slackness averaged across 100 simulation runs.

parameter	$L^{\max}[k]$	P[k]
scenario 1	$\mu \in [4, 6]$	$\mu \in [3, 4.5]$
scenario 2	$\mu \in [1.25, 2.75]$	$\mu \in [1.5, 2.5]$
scenario 3	$\mu \in [4, 6]$	$\mu \in [3, 4.5]$

Table 1. Range specifications for the random variable μ.

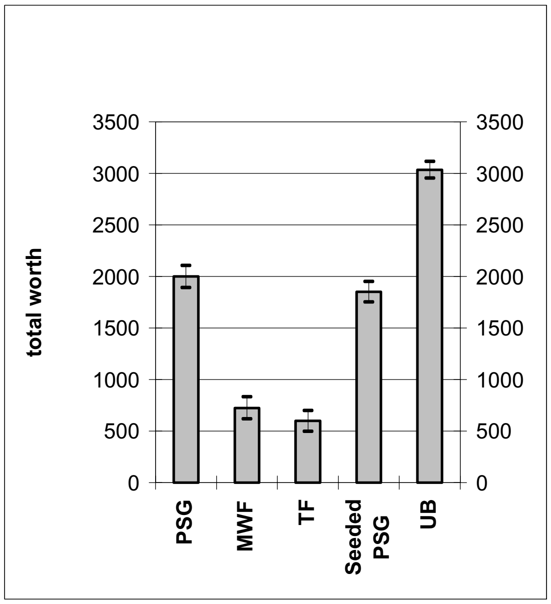

Figure 3. The total worth of allocated strings generated by each heuristic and the upper bound for partial mapping in a highly loaded system (scenario 1).

As the figures show, the PSG and Seeded PSG heuristics perform comparably, providing the best allocations in all three scenarios. It was expected because evolutionary algorithms perform well in large search spaces. Furthermore, the concept of elitism ensured that the algorithms used in these heuristics are globally monotone—i.e., any new solution is either the same as or better than any prior solution.

The MWF and TF heuristics are an attempt to find a solution in one iteration. For this reason, strings considered for mapping are sorted, and then passed in a certain order to the IMR for allocation. Thus, both of these heuristics use just one ordering of strings for generating a resource allocation. In contrast, the PSG heuristics operate with numerous iterations and evaluate numerous orderings of strings. Furthermore, the initial population for the Seeded PSG includes the MWF and TF orderings. This is the reason for relatively poor performance of MWF and TF as compared to that achieved by the PSG heuristics.

When comparing mapping heuristics, the execution time of the heuristics themselves is an important aspect. Both of the fast heuristics (MWF and TF) executed in a few seconds. The evolutionary algorithms (PSG and Seeded PSG) required approximately two hours per single run because they manipulate entire populations and progress through multiple iterations. The LP algorithm from Lingo 9.0 employed for the upper bound calculations runs extremely fast—i.e., its execution time was less than two seconds.

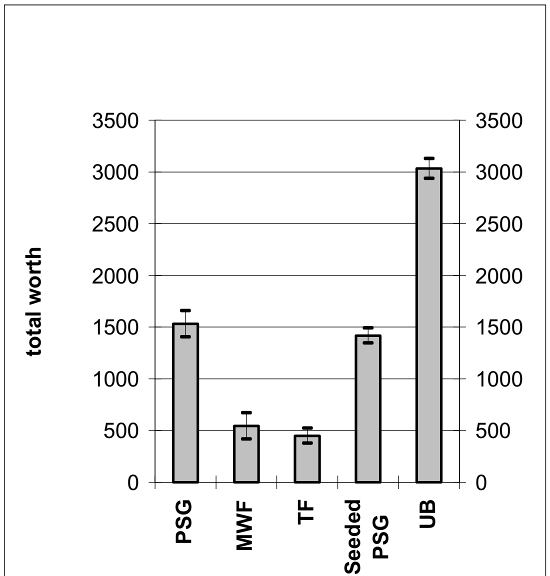

Figure 4: The total worth of allocated strings generated by each heuristic and the upper bound for partial mapping in a QoS-limited system (scenario 2).

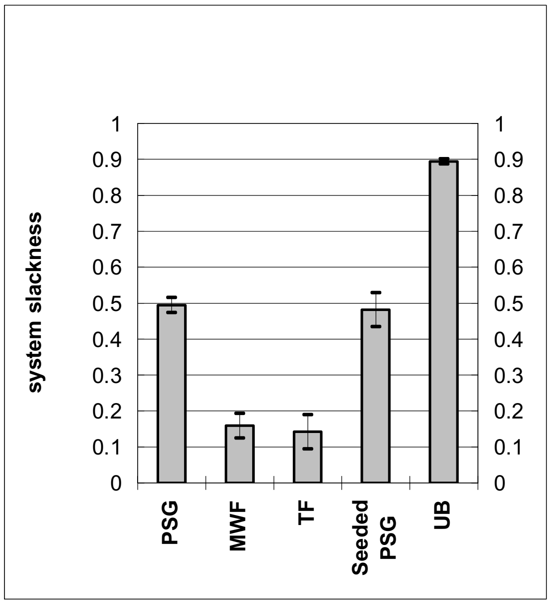

Figure 5: The system slackness generated by each heuristic and the upper bound for complete mapping in a lightly loaded system (scenario 3).

9. Related Work

A number of papers in the literature have studied the issue of initial resource allocation robust against unpredictable workload increases (e.g., [2, 4, 11, 13, 15, 19]). These studies are compared below.

The nature of the problem described in study [2] is similar to the one in this paper. Periodically running applications are organized in sequential strings, which are subject to the imposed end-to-end latency and throughput constraints. That study assumes that the computation time of an application sharing a given machine with $N-1$ other applications is N times its nominal execution time. This results in conservative estimated execution times in a shared environment. Furthermore, there is no notion of nominal utilization—i.e., it is assumed that all applications utilize 100% of the CPU when executing. Our research does not make such assumptions about execution time and CPU utilization; therefore, the approach taken is quite different from that in [2].

Slack-based techniques explored in this work aim to approach robust resource allocation by increasing the amount of unused computation or communication capacity across all hardware resources in the system. A similar performance metric was applied in [13] and [19] to achieve robust schedules in job-shop and real-time environments, respectively. Specifically, an attempt in those works was made to provide each task with extra time (defined as slack) to execute so that some level of uncertainty can be tolerated without having to reallocate.

In [4] it was demonstrated that when application execution parameters are known as a function of workload then a measure of robustness better than system slackness could be used. However, in the TSCE problem, such a function is unknown, and therefore the system slackness is an appropriate measure to use.

The research in [11] 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 one minus the risk of achieving substandard flow time performance. The risk value is calculated by using the approximate distribution of flow time. It is important to note that, in contrast to [11], the workload increases are expected in the TSCE environment but not specified stochastically. If this information was known, the accuracy of a robustness metric could be improved by using techniques similar to those in [11].

Our combination of evolutionary algorithms with the IMR heuristic is conceptually similar to [15, 31]. For example, in [15] the goal is to minimize a weighted combination of the cost of the system and the execution of a set of tasks. A genetic algorithm manipulates a set of chromosomes, where each chromosome composed of a subset of resources available in the system, and an ordering of tasks. A separate greedy heuristic operates on each chromosome to derive a mapping and the associated execution time for the set of tasks.

10. Summary

This paper presents potential methods for efficiently and robustly managing both computation and communication resources in the intended distributed system. The system is expected to operate in an unpredictable environment where the workload is likely to increase, possibly invalidating a resource allocation that was based on the initial workload estimate. In this study, two distinct issues related to static resource allocation are investigated: its feasibility analysis and the heuristic approaches used to determine the allocation. For the former, a two-stage feasibility analysis was developed that allows a scheduler to evaluate a given allocation. The focus in the design of the allocation heuristics was to achieve the highest level of total worth of the strings deployed in the system while maximizing system slackness at that level. System slackness is the measure that quantitatively reflects the system's potential to absorb unpredictable increases in input workload.

The proposed PSG and Seeded PSG heuristics perform well when tested under a variety of simulated computing environments. Essentially, these heuristics combine the GENITOR-based search methods with a specially designed string allocation routine IMR. The comparable performances of these two heuristics indicate their significant potential to produce effective resource allocations in an environment associated with unpredictable workload increases.

References

[1] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. Prasanna, "Greedy heuristics for resource allocation in dynamic distributed heterogeneous computing systems," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2002*), June 2002, pp. 519–530.

[2] S. Ali, J.-K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna, "Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system," *Parallel and Distributed Computing Practices*, Special Issue on Algorithms, Systems and Tools for High Performance Computing, accepted, to appear, http://www.ece.umr.edu/~shoukat/AliShoukatPDCP2004.pdf, accessed September 1, 2004.

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Robust resource allocation for distributed computing systems," *The 2004 International Conference on Parallel Processing (ICPP 2004)*, Aug. 2004.

[4] 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*, Vol. 15, No. 7, July 2004, pp. 630–641.

[5] 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, Vol. 3, No. 3, Nov. 2000, pp. 195–207 (invited).

[6] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments," *16th International Parallel and Distributed Processing Symposium* (*IPDPS'02*), Apr. 2002, pp. 78–85.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of

independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810–837.

[8] E. K. P. Chong and S. H. Żak, *An Introduction to Optimization*, *Second Edition*, John Wiley & Sons Inc., New York, NY, 2001, Chapter 18.

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

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, MA, 1992.

[11] R. L. Daniels and J. E. Carillo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, Vol. 29, No. 11, Aug 1997, pp. 977–985.

[12] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

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

[14] J. L. Devore, *Probability and Statistics for Engineering and Sciences, Fifth Edition,* Duxbury Press, Los Angeles, CA, 1999.

[15] M. K. Dhodhi, I. Ahmad, and R. Storer, "SHEMUS: Synthesis of heterogeneous multiprocessor systems," *Microprocessors and Microsystems*, Vol. 19, No. 6, Aug. 1995, pp. 311–319.

[16] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

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

[18] C. C. Gonzaga, "Path-following methods for linear programming," *SIAM Review*, Vol. 34, No. 2, June 1992, pp. 167–224.

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

[20] J. L. Hennessy, and D. A. Patterson, *Computer Architecture*: *A Quantitative Approach*, *Third Edition,* San Francisco, CA, Morgan Kaufmann, 2003, Chapter 8.

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

[22] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th IEEE Heterogeneous Computing Workshop* (*HCW '95*), Apr. 1995, pp. 30–34.

[23] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280-289.

[24] W. Keuffer, *Visual Coding with Genitor*, Miller Freeman, San Francisco, CA, 1997.

[25] J. K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. S. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, "A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems," *Cluster Computing*, Special Issue on Cluster Computing in Science and Engineering, accepted, to appear.

[26] 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*, Vol. 59, No. 2, Nov. 1999, pp. 107–121.

[27] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, Vol. 39, No. 2, Nov 1987, pp. 117–129.

[28] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, Vol. 38, No. 8, Aug. 1989, pp. 1110–1123.

[29] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Tranaactions, Neural Networks,* Vol. 5, No. 1, Jan. 1994, pp. 96-101.

[30] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 8–22.

[31] J. P. Watson and L. Barbulescu, "Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance," *INFORMS Journal on Computing*, Vol. 14, No. 2, Apr. 2002, pp. 98–123.

[32] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," *3rd International Conference on Genetic*

Algorithms, D. Shaffer, ed., Morgan-Kaufmann, San Francisco, CA, June 1989, pp. 116–121.

Biographies

Vladimir V. Shestak is pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering at Colorado State University, where he has been a Research Assistant since August 2003. His current projects include resource management for clusters for IBM, Boulder. He received his M.S. degree in computer engineering from New Jersey Institute of Technology in May 2003. Prior to joining the New Jersey Institute of Technology he spent three years in industry as a Network Engineer working for CISCO Business Unit in Moscow, Russia. He received his BS degree in electrical engineering from Moscow Engineering Physics Institute, Moscow, Russia. His research interests include resource management within distributed computing systems, algorithm parallelization, and computer network design and optimization.

Edwin K. P. Chong received the B.E.(Hons.) degree with First Class Honors from the University of Adelaide, South Australia, in 1987; and the M.A. and Ph.D. degrees in 1989 and 1991, respectively, both from Princeton University, where he held an IBM Fellowship. He joined the School of Electrical and Computer Engineering at Purdue University in 1991, where he was named a University Faculty Scholar in 1999, and promoted to Full Professor in 2001. Since August 2001, he has been a Professor of Electrical and Computer Engineering, and Professor of Mathematics, at Colorado State University. His current interests are in communication networks and optimization methods. He coauthored the best-selling book, *An Introduction to Optimization, 2nd Edition*, Wiley-Interscience, 2001. He received the NSF CAREER Award in 1995 and the ASEE Frederick Emmons Terman Award in 1998. Professor Chong is a *Fellow* of the IEEE. He was founding chairman of the IEEE Control Systems Society Technical Committee on Discrete Event Systems, and was an IEEE Control Systems Society Distinguished Lecturer. He has been on the editorial board of the *IEEE Transactions on Automatic Control*. He is currently on the editorial board of the journal *Computer Networks*. He has also served on the organizing committees of several international conferences, including the IEEE Conference on Decision and Control, the American Control Conference, the IEEE International Symposium on Intelligent Control, IEEE Symposium on Computers and Communications, and the IEEE Global Telecommunications Conference. He was the Conference (General) Chair for the Conference on Modeling and Design of Wireless Networks, part of PIE ITCom 2001. *An up-to-date vita is available at www.engr.colostate.edu/~echong.*

Anthony A. Maciejewski received the B.S.E.E, M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University under the support of an NSF graduate fellowship. From 1985 to 1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan where he performed work on the development of parallel processing algorithms for computer graphic imaging. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University where he is currently the Head of the Department of Electrical and Computer Engineering. Prof. Maciejewski's primary research interests relate to the analysis, simulation, and control of robotic systems and he has co-authored over 100 published technical articles in these areas. He is an Associate Editor for the *IEEE Transactions on Robotics and Automation*, a Regional Editor for the journal *Intelligent Automation and Soft Computing*, and was co-guest editor for a special issue on "Kinematically Redundant Robots" for the *Journal of Intelligent and Robotic Systems*. He serves on the IEEE Administrative Committee for the Robotics and Automation Society and was the Program Co-Chair (1997) and Chair (2002) for the International Conference on Robotics and Automation, as well as serving as the Chair and on the Program Committee for numerous other conferences. *An up-to-date vita is available at www.engr.colostate.edu/~aam.*

H. J. Siegel holds the endowed chair position of Abell Endowed Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU), where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC). ISTeC a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering at Purdue University. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 300 technical papers. His research interests include heterogeneous parallel and distributed computing, communication networks, parallel algorithms, parallel machine interconnection networks, and reconfigurable parallel computer systems. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and has been on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of six international conferences, and Chair/Co-Chair of five workshops. He is currently on the Steering Committees of three continuing conferences/workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. *An up-to-date vita is available at www.engr.colostate.edu/~hj.*

Lotfi Benmohamed received his Ph.D. from The University of Michigan, Ann Arbor, in 1993. In 1994 he was a visiting researcher at the National Institute of Standards and Technology where he contributed to standards activities within the ATM Forum. In 1995 he joined Bell Laboratories in Holmdel, New Jersey, where he worked on control, design, and management of ATM and IP networks with emphasis on algorithms for congestion control, admission control, and network design. He joined Corvis Corporation as a Senior Network Architect in 2000 where he worked on a number of performance modeling and analysis tasks for optical networking products. Since 2003 he has been with the Johns Hopkins University's Applied Physics Laboratory as a Senior Research Scientist where his current research interests include control and routing in sensor networks and mobile ad-hoc networks.

I-Jeng Wang is a senior research scientist and the sectional supervisor of the probabilistic modeling and inference section with the Research and Technology Development Center at the Johns Hopkins University Applied Physics Laboratory (JHU/APL). He received the M.S. degree from Penn State University in 1991 and the Ph.D. degree from Purdue University in 1996, both in Electrical Engineering. From 1996 to 1997, he was a postdoctoral fellow with the Institute for Systems Research at the University of Maryland, where he conducted research in intelligent control and stochastic approximation. Since October 1997, he has been with JHU/APL where he manages and directs internal research in developing scalable algorithms for solving large-scale DoD problems in areas including resource allocation, wireless networking and pattern recognition. He was the PI of a project on adaptive information control to develop efficient resource allocation techniques for dynamic QoS provisioning over distributed and disparate networks, funded by the DARPA AICE program. He was a Co-PI of a project on autonomous internetworking funded by the Army Collaborative Technology Alliance, under which he leads a multi-organization team to develop scalable and dynamic routing protocols for the Army. He is the Co-PI of a project sponsored by the DARPA IXO ARMS program to develop robust resource management techniques for the Total Ship Computing Environments. His current research interests include stochastic optimization and control, resource allocation, wireless networking, and Bayesian modeling and inference.

Rose Daley is a member of the Senior Professional Staff at the Applied Physics Lab. She holds a B.S. in Electrical Engineering from Rensselaer Polytechnic Institute and an M.S. in Computer Science from the John Hopkins University, specializing in Distributing Computing. She has over seventeen years experience architecting and implementing software systems, including both distributed large-scale tactical systems encompassing multiple operating systems and communication protocols, and enterprise systems with large databases on internal Intranets. She is the Architecture lead for Mission-Centric Network Defense (MCND) IR&D effort, an approach to determining tactical mission sensitivity to network resource attacks and casualties, the system architect and lead software engineer for the Active Adjunct Processor for the AN/SQQ-89 surface ship sonar system (a distributed computing system for complex sonar detection/classification algorithms), and senior software engineer for the Tactical Combat Training System (TCTS), the AN/BSY-2 Team Trainer, and the CCS MK2 Submarine Combat System.