ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

J. Parallel Distrib. Comput. ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

A hybrid Branch-and-Bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems☆

Vladimir Shestak[a], Edwin K.P. Chong[a,c], Howard Jay Siegel[a,b,*], Anthony A. Maciejewski[a], Lotfi Benmohamed[d], I-Jeng Wang[d], Rose Daley[d]

[a]*Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523–1373, USA*
[b]*Department of Computer Science, Colorado State University, Fort Collins, CO 80523–1373, USA*
[c]*Department of Mathematics, Colorado State University, Fort Collins, CO 80523–1373, USA*
[d]*Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD 20723-6099, USA*

Received 1 November 2006; received in revised form 8 May 2007; accepted 22 May 2007

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 shipboard 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 strings, each made up of an ordered sequence of applications. There is a quality of service (QoS) minimum throughput constraint that must be satisfied for each application in a string, and a maximum utilization constraint that must be satisfied on each of the hardware resources in the system. The challenge, therefore, is to efficiently and robustly manage both computation and communication resources in this unpredictable environment to achieve high performance while satisfying the imposed constraints. This work addresses the problem of finding a robust initial allocation of resources to strings of applications that is able to absorb some level of unknown input workload increase without rescheduling. The proposed hybrid two-stage method of finding a near-optimal allocation of resources incorporates two specially designed mapping techniques: (1) the Permutation Space Genitor-Based heuristic, and (2) the follow-up Branch-and-Bound heuristic based on an Integer Linear Programming (ILP) problem formulation. The performance of the proposed resource allocation method is evaluated under different simulation scenarios and compared to an iteratively computed upper bound.
© 2007 Published by Elsevier Inc.

Keywords: Heterogeneous systems; Scheduling; Heuristic methods; Evolutionary computing and genetic algorithms; Constrained optimization; Integer programming

1. Introduction and problem statement

Often, parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of quality of service (QoS). This reality has inspired an increasing interest in robust design. The following are some examples, as cited in [2]. The Robust Network Infrastructures Group at the Computer Science and Artificial Intelligence Laboratory at MIT takes the position that "... a key challenge is to ensure that the network can be robust in the face of failures, time-varying load, and various errors". The research at the User-Centered Robust Mobile Computing Project at Stanford "concerns the hardening of the network and software infrastructure to make it highly robust". The Workshop

☆ This research was supported by NSF under Grant no. CNS-0615170, by the DARPA Information Exploitation Office under Contract no. NBCHC030137, by the IBM Ph.D. Fellowship Program, 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), and by the Colorado State University George T. Abell Endowment. Approved for public release, distribution unlimited.

* Corresponding author. Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523–1373, USA.
E-mail addresses: shestak@engr.colostate.edu (V. Shestak), echong@engr.colostate.edu (E.K.P. Chong), hj@engr.colostate.edu (H.J. Siegel), aam@engr.colostate.edu (A.A. Maciejewski), lotfi.benmohamed@jhuapl.edu (L. Benmohamed), i-jeng.wang@jhuapl.edu (I.-J. Wang), rose.daley@jhuapl.edu (R. Daley).

0743-7315/$ - see front matter © 2007 Published by Elsevier Inc.
doi:10.1016/j.jpdc.2007.05.011

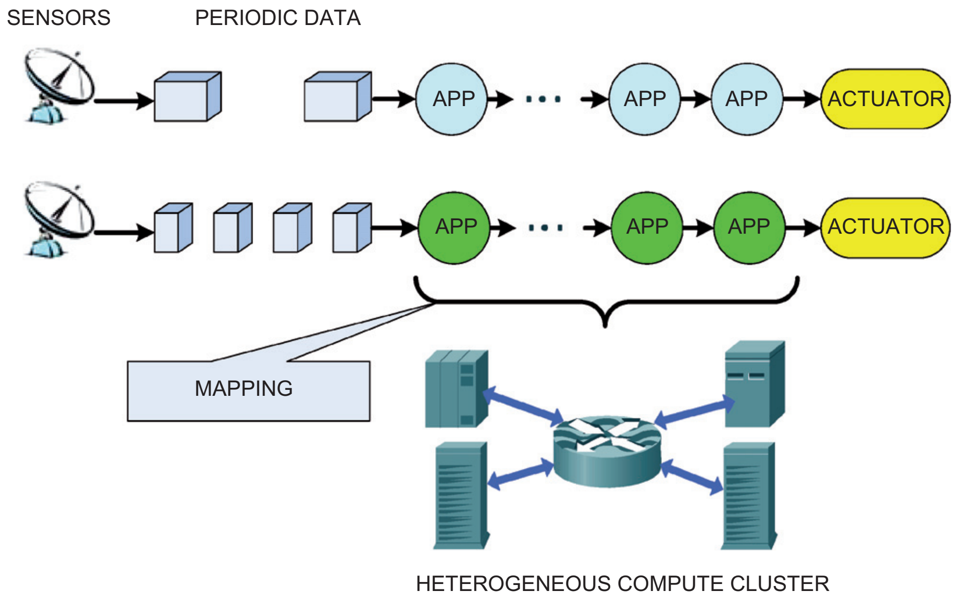


Fig. 1. The considered part of the Adaptive and Reflective Middleware Systems shipboard environment. Each sensor generates data periodically and forwards it to the distributed heterogeneous processing system. Multiple applications organized in strings must process data to satisfy the imposed QoS constraints. Thus, a resource allocation becomes a very important issue in the system.

on Large-Scale Engineering Networks: Robustness, Verifiability, and Convergence (2002) concluded that the "Issues are . . .. being able to quantify and design for robustness . . . ." There are many other projects of similar nature at other organizations.

This research investigates the problem of robust static resource allocation for shipboard computing resources in the Adaptive and Reflective Middleware Systems program supported by the DARPA Information Exploitation Office [1]. In this paper, a resource allocation problem is addressed for a part of the proposed shipboard environment, depicted schematically in Fig. 1, where a limited subset of the Adaptive and Reflective Middleware Systems application model is considered. As Fig. 1 shows, the target system consists of a number of sensors generating raw data forwarded to the heterogeneous distributed computing system for processing. The computing system itself is composed of a set of machines of various types, a communication network, and continuously running applications processing data coming from the sensors. Data processing in the system must be done by a sequence of applications in a pipeline fashion; this requirement imposes a QoS constraint on each application's processing time and each internal data transfer's time between applications.

The system is configured with an initial *mapping* (i.e., an allocation of computing and networking resources to applications) that is used when the system is first put into operation. The system is expected to function in an uncertain physical environment where the workload, i.e., the load presented by a set of sensors, is likely to change unpredictably over time, possibly causing a QoS violation. When this occurs, resources in the system need to be reallocated reactively, which results in a temporal performance degradation and, thus, is highly undesirable. Therefore, the general focus of this work is on developing a resource allocation technique to determine a *robust* mapping capable of absorbing the maximum increase in workload without a run-time reallocation of resources. Reallocation techniques and dynamic mapping are outside the scope of this paper, but a variety of them can be found in the literature (e.g., [21,36,41,42]).

Specifically, two resource allocation scenarios are addressed in this paper that differ in their performance goals. A partial resource allocation scenario occurs in an *oversubscribed* system where one or more sequences of applications considered for mapping cannot be allocated due to the limited system resources or predicted QoS constraint violations. Given such a scenario, the primary performance goal for a mapper is to find a static (i.e., one found during an off-line planning phase) initial mapping maximizing a "total worth" of the workload processed. In contrast, a complete resource allocation scenario is relevant for a system that has enough resources to accommodate all applications considered without violating any of the imposed QoS constraints. In this case, a "system slackness" metric reflecting the system's capacity to absorb workload surges is a major optimization criterion for a static mapping.

Given that the problem of resource allocation in distributed systems is NP-complete (e.g., [9,24]), the development of heuristic techniques to find near-optimal solutions became an active area of research (e.g., [3,5,6,17,30,35,44]). In general, there are two major classes of resource allocation approaches widely used in practice: greedy heuristics and global optimization algorithms. Unlike rather time-consuming global optimization algorithms, greedy heuristics are relatively fast in generating a single solution; this feature often makes them an appropriate choice to use in dynamic (i.e., on-line mapping) systems. However, the quality of solutions based on greedy heuristics is usually lower than that produced by global optimization algorithms that progressively iterate through multiple solutions.

One type of global optimization algorithms is a class of evolutionary techniques. In principle, such techniques rely on an "intelligent" randomized search where a solution is picked in the solution space, and its fitness is then evaluated. The efficiency of this method often suffers when applied to constrained optimization problems because of the time wasted generating infeasible solutions. To resolve this issue, the paper proposes an approach in the first stage of finding a resource allocation where the search space of the evolutionary algorithm differs from the actual solution space, and a specifically constructed greedy heuristic is used to link these two spaces.

Another drawback of evolutionary algorithms is a lack of structural organization of the underlying search process. For problems of realistic size, it is highly unlikely that randomized search will find a global optimal solution in a reasonable amount of time. Furthermore, even if an algorithm converges to the global optimal solution it has no means of proving it. To overcome this problem, a hybrid two-stage approach in this work proposes that the final solution of the evolutionary algorithm is passed to a Branch-and-Bound technique based on an Integer Linear Programming formulation of the resource allocation problem. The key point is that the well-structured tree search in the Branch-and-Bound algorithms becomes significantly more efficient when a high-quality solution is received that can be used for pruning the search tree. Furthermore, a backtracking mechanism in our second-stage Branch-and-Bound algorithm allows the upper bound on performance metrics to be tightened as the algorithm progresses.

In addition to the proposed hybrid two-stage approach to resource allocation utilizing a combination of the evolutionary and greedy heuristics in the first stage and a Branch-and-Bound algorithm in the second stage, the contributions of this work include developing the application and hardware models of the considered part of the shipboard environment, quantifying the performance goals for different scenarios, evaluating the relative performance of the heuristics developed, and deriving mathematical bounds on performance based on a Linear Programming relaxation method.

The remainder of this paper is organized in the following manner. Section 2 develops models for the workload and hardware platform. Section 3 presents a quantitative basis for the performance measure for a given resource allocation. In Section 4, the Genitor-based evolutionary algorithm is described along with a special-purpose greedy mapping routine developed to generate solutions for both allocation scenarios in the first stage. A mathematical model for finding performance upper bounds in different scenarios based on a Linear Programming relaxation is provided in Section 5. Section 6 presents a set of Branch-and-Bound algorithms developed to improve on the first stage resource allocations and tighten the upper bounds. The simulation setup, results, and performance evaluation of the heuristics are discussed in Section 7. A sampling of some related work is presented in Section 8. Section 9 concludes the paper. A glossary of notation used in the paper is tabulated in Appendix A.

2. System model

The considered distributed system is composed of a number of heterogeneous computational resources distributed across a shipboard environment and connected by a communication network.

The functionality of the communication network is modeled by all possible independent virtual point-to-point communication routes, each characterized by a maximum available bandwidth. Existing networking technologies can enforce this 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 given shipboard environment, a string is defined as a continuously executing sequence of applications connected in precedence order by specified data transfers. Data are received by an application from the preceding application, or from a sensor that generates data sets with a fixed period. The output produced by the string serves as an input to other applications or to actuators.

Let S^k be the kth string, specified by a sequence of n_k applications: $S^k = a_1^k a_2^k \cdots a_{n_k}^k$. To model the importance of each string in the system, for each k, the kth string is preassigned one of three possible worth factors, $W[k] \in \{1, 2, 3\}$. Worth factors associated with application strings in this work were needed to prioritize different missions carried out on a ship. Because the functionality of each mission is supported by a specific application string (or a set of application strings), priorities allow the resource allocation algorithm to select the most valuable missions, especially when the system has limited resources. Our assignment of worth factors in this work is based on expected characteristics of applications in the Adaptive and Reflective Middleware Systems project. For any practical implementation, these factors must be readjusted in a system-dependent way. Typically in a military environment worth factors may be set by the command hierarchy.

Let $P[k]$ be the period associated with string S^k, where each a_i^k must execute once each period. The minimum throughput QoS constraint requires that the computation time of any application or the time of any inter-application data transfer in S^k be no larger than $P[k]$. Such an enforcement allows each string to process data in a *pipeline* fashion resulting in high processing efficiency for the entire system. Assuming that a resource allocation for string S^k is made, let $m[i, k]$ denote the machine to which application a_i^k is assigned. Let $t_{\text{comp}}^k[i]$ be the estimated computation time for application a_i^k for each data set (executing on $m[i, k]$). Let $t_{\text{tran}}^k[i]$ be the estimated transfer time required to send the 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.

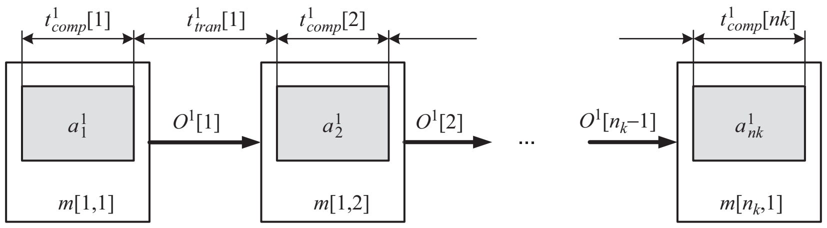


Fig. 2. The string model for string 1. Shaded rectangles denote applications in the string while white rectangles represent the machines where these applications are executed. The arrows represent output data transfers within the string.

A typical allocation of string S^k in the system is illustrated in Fig. 2 below.

Mathematically, for a given resource allocation for a string, the aforementioned minimum throughput QoS constraint is satisfied if

$$\begin{cases} t^k_{\text{comp}}[i] \leqslant P[k], & 1 \leqslant i \leqslant n_k, \\ t^k_{\text{tran}}[i] \leqslant P[k], & 1 \leqslant i \leqslant n_k - 1. \end{cases} \tag{1}$$

If all conditions in (1) hold for a given allocation, the allocation is said to be feasible with respect to the minimum throughput QoS constraint. Because both machines and communication routes are assumed to be shared, $t^k_{\text{comp}}[i]$ and $t^k_{\text{tran}}[i]$ 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. In addition to the minimum throughput QoS constraint imposed on strings, the overall utilization of each computation or communication resource must not exceed its full capacity when the system is loaded. Evaluating that the utilization and minimum throughput QoS constraints are satisfied is integrated into the mapping techniques presented in the following sections.

Two parameters are used in the given shipboard 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 $\bar{t}^k[i, j]$ is the time required by application a^k_i in string S^k to process a *nominal* data set on machine j running in non-multitasking mode. A nominal data set is a data set of mean complexity which is determined based on past executions. Due to the multitasking environment, $t^k_{\text{comp}}[i] \geqslant \bar{t}^k[i, m[i, k]]$. The nominal CPU utilization $\bar{u}^k[i, j]$ is the average CPU utilization of machine j when a^k_i executes its nominal data set. The product $\bar{t}^k[i, j] \times \bar{u}^k[i, j]$ can be interpreted as the fixed amount of CPU work required for application a^k_i to process a nominal 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:

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

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

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

The term $\bar{t}^k[i, j] \times \bar{u}^k[i, j]/P[k]$ represents the average CPU utilization allocated for application a^k_i over $P[k]$. It is important to note that this is the *minimum* required average CPU utilization that allows a^k_i to complete processing without a throughput QoS constraint violation. Recall from (1) that a data set processing time of each application in string S^k must be less than or equal to $P[k]$.

The sum of such minimum CPU utilizations across all the applications executing on j determines the overall machine utilization.

If $b[j_1, j_2]$ denotes the time required to transmit one bit of data over the communication route from machine j_1 to machine j_2 then the overall communication route utilization $U^{\text{route}}[j_1, j_2]$ is

$$U^{\text{route}}[j_1, j_2] = b[j_1, j_2] \times \sum_{k=1}^{A} \sum_{i=1}^{n_k-1} \left(\frac{O^k[i]}{P[k]} \times \mathbf{1}(m[i, k] = j_1 \text{ and } m[i+1, k] = j_2) \right). \tag{3}$$

The term $O^k[i]/P[k]$ 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 minimum throughput QoS constraint violation.

For a given allocation, if the utilization values computed with (2) and (3) are not greater than one for each machine and communication route then the system is considered to be operating in a feasible (not overloaded) mode.

3. 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. The secondary component is system slackness. Assuming M machines in the system, let system slackness Λ be a measure of the minimum utilization capacity remaining across all computation and communication resources:

$$\begin{cases} A = \max\{U^{\text{machine}}[j] : 1 \leqslant j \leqslant M\}, \\ B = \max\{U^{\text{route}}[j_1, j_2] : 1 \leqslant j_1, j_2 \leqslant M\}, \\ \Lambda = 1 - \max\{A, B\}. \end{cases} \quad (4)$$

Component A in Eq. (4) identifies the machine with the maximum utilization. Similarly, component B indentifies the communication route with the maximum utilization. Thus, [$1-$ maximum over A and B], i.e., system slackness Λ, shows that all resources in the system have at least Λ unutilized capacity. The system slackness is based on a rather intuitive concept—it quantifies unutilized capacity of the resource with the highest utilization because this resource is a potential bottleneck in a distributed system. A similar approach was explored earlier in the field, e.g., [3,12]. For example, the study in [12] uses slack-based techniques for producing robust resource allocations in a job-shop environment. Specifically, an attempt in that work 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.

According to [4], a resource allocation is defined to be robust with respect to specified system performance features against uncertainties in specified system parameters if degradation in these features is limited. In this work, the system is considered robust if it is able to absorb limited unpredictable changes in input workload which increase resource utilizations without revising a given resource allocation. System slackness is used as a quantitative measure of robustness. The goal of the mapping heuristics developed in this research is to achieve the highest level for the primary component, and then maximizing system slackness Λ at that level.

With the given "worth" scheme, a high-worth string has the same value as three low-worth strings. A different, alternate scheme is possible, where a high-worth string has 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 [27], is outside the current requirements of this work.

4. Basic evolutionary mapping algorithm

4.1. Overview

This section presents an evolutionary mapping algorithm used as a basis for the problem of finding an initial static mapping in the complete and partial allocation scenarios. To explain the main idea used in the algorithm's development, the following notation needs to be introduced. Let the permutation space be all possible orderings of the strings considered for mapping, and let the solution space be all possible application-to-machine assignments. Recall that an allocation is considered feasible for deployment if none of the computation or communication hardware resources is loaded beyond its maximum utilization capacity.

It was observed experimentally that the straightforward implementations of evolutionary algorithms, e.g., a genetic algorithm [44], operating in the solution space, failed to find any feasible allocation even for a relatively small set of strings in a reasonable amount of time (5 h in our experiments). This phenomenon can easily be explained by the random-search-based principle utilized in evolutionary algorithms. Random application-to-machine assignments generated in the solution space resulted in too many applications mapped on a single machine or communication route eventually violating the QoS constraint.

Therefore, the Genitor-based evolutionary algorithm presented in this section was modified to search over the permutation space instead of directly over the solution space. An ordering of strings in the permutation space is translated into a mapping in the solution space by repetitively applying the Incremental Mapping Routine described below. The Incremental Mapping Routine is designed to map a single string; different allocations are achieved when different orderings of strings are sequentially processed by the routine.

Our choice of the Genitor-based evolutionary algorithm was based on high performance results that this algorithm evidenced in many problem domains related to resource allocation in distributed systems (e.g., [15,43]). Furthermore, based on our previous experiments, the convergence rate of Genitor-based algorithms is usually higher than that of other modern evolutionary heuristics, e.g., Simulated Annealing [31], Ant Colony Optimization [16].

4.2. Incremental mapping routine

The allocation algorithm used in the Incremental Mapping Routine is based on a greedy mapping technique. This routine 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 (on average), determined in step 1 of the pseudo code shown below, the heuristic maps all the intermediate applications along the string up to the next most computationally intensive application (on average). In selecting a mapping, a parameter of interest is the maximum value of the resource utilizations (given by Eqs. (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, 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 Incremental Mapping Routine approach attempts to map computationally intensive applications early, but also maps their neighboring applications, so that network utilization is taken into account as the heuristic progresses.

To describe the Incremental Mapping Routine in detail some additional notation must be introduced. Let $U^{\text{machine}}[j, i, k]$ be the utilization of machine j if application a_i^k were assigned to machine j (in addition to the applications assigned previously to this machine). Similarly, let $U^{\text{route}}[j_1, j_2, i, k]$ be the utilization of the communication route if application a_i^k were assigned to machine j_1 and passed its output to its successor mapped on machine j_2. Let the average nominal execution time $\bar{t}_{\text{av}}^k[i]$ for application a_i^k be given as $\bar{t}_{\text{av}}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \bar{t}^k[i, j]$, and let the average nominal machine CPU utilization $\bar{u}_{\text{av}}^k[i]$ for application a_i^k be given as $\bar{u}_{\text{av}}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \bar{u}^k[i, j]$. A detailed description of the Incremental Mapping Routine follows.

(1) As a starting point, identify application $a_{i_{\max}}^k$ in the given string S^k as follows:

$$i_{\max} = \underset{i=1,\dots,n_k}{\arg\max} \left\{ \frac{\bar{t}_{\text{av}}^k[i] \times \bar{u}_{\text{av}}^k[i]}{P[k]} \right\}.$$

(2) **If** $1 < \min_{j=1,\dots,M} \left\{ U^{\text{machine}}[j, i_{\max}, k] \right\}$
return mapping failed.

(3) Assign application $a_{i_{\max}}^k$ to the machine $m_{i_{\max},k}$ found as

$$m[i_{\max}, k] = \underset{j=1,\dots,M}{\arg\min} \left\{ U^{\text{machine}}[j, i_{\max}, k] \right\}.$$

(4) Initialize set $D = \left\{ a_{i_{\max}}^k \right\}$.

(5) **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 a new unassigned application $a_{i_{\max}}^k$ in the given string S^k as follows:

$$i_{\max} = \underset{i=1,\dots,n_k \text{ and } a_i^k \notin D}{\arg\max} \left\{ \frac{\bar{t}_{\text{av}}^k[i] \times \bar{u}_{\text{av}}^k[i]}{P[k]} \right\};$$

(c) **while** $i_{\max} > i_{\text{right}}$ **do**

- $i_{\text{right}} = i_{\text{right}} + 1$;
- **if** $1 < \min_{j=1,\dots,M} \left[\max \left\{ U^{\text{machine}}\left[j, i_{\text{right}}, k\right], U^{\text{route}}\left[m\left[i_{\text{right}} - 1, k\right], j, i_{\text{right}}, k\right] \right\} \right]$
return mapping failed;
- assign to the machine found as follows:

$$m\left[i_{\text{right}}, k\right] = \underset{j=1,\dots,M}{\arg\min} \left[\max \left\{ U^{\text{machine}}\left[j, i_{\text{right}}, k\right], U^{\text{route}}\left[m\left[i_{\text{right}}-1, k\right], j, i_{\text{right}}, k\right] \right\} \right];$$

- insert application $a_{i_{\text{right}}}^k$ in set D;

(d) **while** $i_{\max} < i_{\text{left}}$ **do**

- $i_{\text{left}} = i_{\text{left}} - 1$;
- **if** $1 < \min_{j=1,\dots,M} \left[\max \left\{ U^{\text{machine}}[j, i_{\text{left}}, k], U^{\text{route}}[j, m[i_{\text{left}} + 1, k], i_{\text{left}}, k] \right\} \right]$
return mapping failed;
- assign $a_{i_{\text{left}}}^k$ to the machine $m[i_{\text{left}}, k]$ found as follows:

$$m[i_{\text{left}}, k] = \underset{j=1,\dots,M}{\arg\min} \left[\max \left\{ U^{\text{machine}}[j, i_{\text{left}}, k], U^{\text{route}}[j, m[i_{\text{left}} + 1, k], i_{\text{left}}, k] \right\} \right];$$

insert application $a_{i_{\text{left}}}^k$ in set D.

4.3. Permutation Space Genitor-Based heuristic

The Permutation Space Genitor-Based heuristic was developed by combining the Incremental Mapping Routine 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., [6,26,38,46]). Designed for a given resource allocation 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 Incremental Mapping Routine.

If the mapping produced by the Incremental Mapping Routine fails for a string due to a utilization violation, then the string is skipped, and the routine proceeds with the next string. The two-component performance metric is used to measure the fitness of each chromosome. Recall from Section 3 that the primary component of the performance metric indicates total worth of the strings allocated in the system while the secondary component indicates system slackness.

The Permutation Space Genitor-Based heuristic was implemented as follows. First, an initial population is generated randomly by reordering the initial set of strings. A population size of 250 chromosomes was determined experimentally for a given setup; any further increase in the size of the population does not improve the performance of the heuristic. After a mapping involving the Incremental Mapping Routine, the entire set of chromosomes is sorted (ranked) by their fitness (system slackness for the complete allocation scenario; total worth for the partial allocation scenario with ties broken by system slackness). 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. Each offspring is then inserted in the ordered population, and the worst two chromosomes are dropped.

In the crossover step, for the selected pair of parent chromosomes a random cut-off point is generated that divides the chromosomes into top and bottom parts. Next, the strings in each of the top parts are reordered. The new ordering of the strings in one top part corresponds to the relative positions of its 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.

After each crossover, the same special function (described below) is applied to select a chromosome for mutation. The 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 an offspring generated by crossover.

The special function for selecting parent chromosome(s) is a bias function, used to provide a specific selective pressure [46]. 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. Experimentally, by varying the bias values across the range [1,2] in steps of 0.1, the best bias for this system was found to be 1.6.

As the Permutation Space Genitor-Based heuristic runs, the crossover operator will be iteratively repeated followed by the mutation operator until one of the stopping conditions is reached: (1) 120 min to execute, (2) 2000 iterations without a change in the best chromosome, or (3) either the mutation or the crossover operator failed to produce a never before examined chromosome within 10 min.

The developed Permutation Space Genitor-Based heuristic was used in the conducted experiments in the following ways:

- In each run of the complete resource allocation scenario, the Permutation Space Genitor-Based heuristic was applied to compute the complete initial allocation while maximizing the secondary component of the objective metric, i.e., system slackness. This baseline solution was then used in the follow-up Branch-and-Bound algorithm (described in Section 6) to reduce the search space.
- In the partial resource allocation scenario, Permutation Space Genitor-Based heuristic was used to find a subset of mapped strings that results in the maximized total worth, using system slackness to break ties. The determined subset was then passed to the Branch-and-Bound heuristic, which aimed to improve an allocation for the subset with respect to system slackness.

5. Integer Linear Programming formulation

5.1. Overview

An Integer Linear Programming form and a corresponding Linear Programming form [11,33] are derived in this section for each allocation scenario. The Integer Linear Programming form fully describes the optimization problem considered in each scenario while its relaxation into the Linear Programming form: (1) provides the initial upper bound on the performance metric, and (2) establishes a basis for a node selection in the Branch-and-Bound algorithm presented in the following section.

Let $\boldsymbol{c} \in \mathbb{R}^{\alpha}$ and $\boldsymbol{b} \in \mathbb{R}^{\beta}$ be real vectors, and $\boldsymbol{\Upsilon} \in \mathbb{R}^{\beta\times\alpha}$ be a real matrix. If $\boldsymbol{h}$ is a vector composed of α decision variables [8] then the canonical Integer Linear Programming formulation is written as

$$\text{maximize } Z_{\text{ILP}} = \boldsymbol{c}^{\mathrm{T}} \times \boldsymbol{h}; \quad \text{subject to (I) } (\boldsymbol{\Upsilon} \times \boldsymbol{h} \leqslant \boldsymbol{b}, \text{ (II) } \boldsymbol{h} \text{ consists of integers.} \tag{5}$$

Constraint (II) makes the Integer Linear Programming problem NP-complete [32]. If this constraint is ignored, i.e., $\boldsymbol{h} \in \mathbb{R}^{\alpha}$ then the Integer Linear Programming form is relaxed into an Linear Programming form. The global optimal solution for the Linear Programming form, which is the upper bound for the Integer Linear Programming form, can be found in *polynomial* time, e.g., by applying one of the interior-points methods [20].

Let the binary decision variable $x[i, k, j]$ be equal to 1 if application a_i^k is assigned to machine j and equal to 0 if a_i^k is not assigned to machine j. Similarly, let $y[i, k, j_1, j_2]$ be equal to 1 if the output generated by a_i^k is transferred over the communication route from machine j_1 to machine j_2 and 0 if it is not transferred over that communication route. Due to the new variables, Eq. (2) for machine utilization and Eq. (3) for communication route utilization need to be restated:

$$U^{\text{machine}}[j] = \sum_{k=1}^{A} \sum_{i=1}^{n_k} \left(\frac{\bar{t}^k[i, j]}{P[k]} \times \bar{u}^k[i, j] \times x[i, k, j] \right), \tag{6}$$

$$U^{\text{route}}[j_1, j_2] = b[j_1, j_2] \times \sum_{k=1}^{A} \sum_{i=1}^{n_k-1} \left(\frac{O^k[i]}{P[k]} \times y[i, k, j_1, j_2] \right). \tag{7}$$

5.2. Complete allocation scenario

In the complete resource allocation scenario, the objective is to *maximize* system slackness Λ given by (4), because all mappings would have the same total worth. Thus, the objective function for the Integer Linear Programming form is formally stated as

$$\text{maximize } \Lambda = \min\left(\left\{ 1 - U^{\text{machine}}[j] : j \in M \right\} \cup \left\{ 1 - U^{\text{route}}[j_1, j_2] : j_1, j_2 \in M \right\} \right). \tag{8}$$

Suppose that Q represents the total number of strings considered for mapping in the system. The objective function is subject to set of conditions (a)–(f), explained in detail below:

$$[i,k,j] \in \{0,1\} \quad \text{for } 1 \leqslant i \leqslant n_k,\ \ 1 \leqslant k \leqslant Q,\ \ 1 \leqslant j \leqslant M; \tag{a}$$

$$\sum_{j=1}^{M} x[i,k,j] = 1 \quad \text{for } 1 \leqslant i \leqslant n_k,\ \ 1 \leqslant k \leqslant Q; \tag{b}$$

$$x[i,k,j_1] = \sum_{j_2=1}^{M} y[i,k,j_1,j_2] \quad \text{for } 1 \leqslant i \leqslant n_k - 1,\ \ 1 \leqslant k \leqslant Q,\ \ 1 \leqslant j_1 \leqslant M; \tag{c}$$

$$x[i,k,j_2] = \sum_{j_1=1}^{M} y[i,k,j_1,j_2] \quad \text{for } 2 \leqslant i \leqslant n_k,\ \ 1 \leqslant k \leqslant Q,\ \ 1 \leqslant j_2 \leqslant M; \tag{d}$$

$$U^{\text{machine}}[j] \leqslant 1 \quad \text{for } 1 \leqslant j \leqslant M; \tag{e}$$

$$U^{\text{route}}[j_1, j_2] \leqslant 1 \quad \text{for } 1 \leqslant j_1, j_2 \leqslant M. \tag{f}$$

Condition (a) explicitly restricts decision variables $x[i,k,j]$ to integer binaries $\{0,1\}$ corresponding to the "assigned/not assigned" allocation choice for application a_i^k on machine j. Condition (b) forces each application to be mapped to the system. Conditions (c) and (d) link the communication route assignment of output $O^k[i]$ generated by application a_i^k to the allocation of applications a_i^k and a_{i+1}^k. The enforcement of utilization feasibility in the system is represented by the remaining two conditions (e) and (f). The objective function (8) and conditions (e) and (f) are based on Eqs. (6) and (7). The binary restriction on decision variables $y[i,k,j_1,j_2]$ is imposed implicitly by conditions (a), (c), and (d).

The objective function (8) and the set of conditions (a)–(f) formulate an optimization problem in the complete allocation scenario in the Integer Linear Programming form. The Linear Programming form that provides an upper bound on system slackness follows from the Integer Linear Programming form as the decision variables in condition (a) are relaxed to real numbers, i.e., for $1 \leqslant i \leqslant n_k$, $1 \leqslant k \leqslant Q$, $1 \leqslant j \leqslant M$, $0 \leqslant x[i,k,j] \leqslant 1$. This implies that each $y[i,k,j_1,j_2]$ is also a real number.

5.3. *Partial allocation scenario*

In the partial resource allocation scenario the primary objective is to maximize the total worth of the strings deployed in the system, as defined in Section 3. This transforms into the formal representation of an objective function in the Integer Linear Programming form:

$$\text{maximize} \sum_{k=1}^{Q} \sum_{i=1}^{n_k} \left(W[k] \times \sum_{j=1}^{M} x[i,k,j] \right). \tag{9}$$

The objective function is subject to conditions (a)–(f), where condition (b) needs to be restated due to the limited computation or communication capacity of the resources available in the partial allocation scenario:

$$\sum_{j=1}^{M} x[i,k,j] \in \{0,1\} \quad \text{for } 1 \leqslant i \leqslant n_k,\ \ 1 \leqslant k \leqslant Q. \tag{b$'$}$$

The modified condition (b′), along with conditions (a), (c), and (d), requires each of Q strings to be either completely mapped or not mapped at all, precluding cases where the number of mapped applications in the string is less than n_k. An Linear Programming form that provides an upper bound on total worth in the partial allocation scenario is obtained from the derived Integer Linear Programming form when conditions (a) and (b′) are relaxed to real numbers confined to the interval [0,1].

The Linear Programming forms presented above result in the initial upper bounds on system slackness and total worth in the complete and partial allocation scenarios, respectively. However, tighter upper bounds were achieved iteratively in both scenarios by applying the developed Branch-and-Bound algorithms described in the next section.

6. Branch-and-Bound heuristics

Due to the NP-complete nature of the Integer Linear Programming problem, in general its global optimal solution cannot be found in polynomial time except for some special cases described in the literature (e.g., [7,33,47]). In the complete allocation scenario, the Branch-and-Bound algorithm, presented in this section, was designed to improve a suboptimal solution produced by the Permutation Space Genitor-Based algorithm and to tighten the initial upper bound on system slackness. In the partial allocation scenario, the Branch-and-Bound algorithm was developed to tighten the upper bound on total worth.

6.1. *Complete allocation scenario*

The proposed Branch-and-Bound algorithm is a tree search beginning at a root node that is a null solution. In the entire tree, interior nodes represent intermediate solutions (a subset of applications are assigned to machines), and leaf nodes represent leaf solutions (all applications are assigned to machines). The intermediate solution of a child node has one more application mapped than its parent node. Call this additional application a. Each parent node expands into M children, one for each possible mapping of a. Nodes are said to be open until they are expanded, whereupon they become closed.

The intermediate solution at each node is characterized by a value of the secondary component of the objective function (system slackness) found by solving the Linear Programming form, derived in Section 5, for this scenario. When solving this form, the decision variables that correspond to applications already mapped are set to binary integers $\{0,1\}$ according to application assignments; other decision variables are relaxed to real numbers. The Linear Programming solutions at nodes are used as bounds to curtail the search. Specifically, a node is pruned (closed) if one of the following two conditions holds:

(I) no solution can be found for the Linear Programming form at a given node, i.e., no solution can be found that satisfies the set of constraints (a)–(g); (II) the value of the objective function found for the Linear Programming form is not greater than the highest value of the objective function among the known leaf solutions.

As it follows from condition (II), a known high-quality leaf solution helps to avoid the explicit examination of many early nodes in the tree and, thus, significantly narrows the search space. In the proposed Branch-and-Bound algorithm, the baseline solution generated by the developed Permutation Space Genitor-Based heuristic is used for pruning until the Branch-and-Bound algorithm determines a better solution from a leaf node. Although pruning helps to limit the search space, the problem of finding the global optimal Integer Linear Programming solution in the complete allocation scenario remains quite time-consuming due to the large solution space composed of numerous application-to-machine assignment combinations. Therefore, the total execution time for the Branch-and-Bound heuristic was limited to 5 h.

The Branch-and-Bound algorithm can be summarized by the following procedure. Starting from the root node, the Branch-and-Bound heuristic iteratively attempts to reach the bottom level of the tree by selecting a new parent from among the open children resultant from the previous expansion. The child selected is the one with the maximum objective function value. Such a node expansion method is referred in the literature as a depth-first search [33]. When the bottom level of the tree is reached, M leaf solutions are generated. If the best of these leaf solutions has an objective function value higher than that used for pruning before, this new leaf solution is now the overall best found so far, and will be used for future pruning. Furthermore, this leaf solution is applied to evaluate all open intermediate nodes currently included in the tree to close the nodes that satisfy condition (II). If none of the nodes considered for picking a parent node is open or the bottom level of the tree is reached, a new startup node is selected in the tree to continue the search process. The startup node selection, called backtracking [33], is based on the highest objective function value among the Linear Programming solutions associated with all *open* interior nodes currently included in the tree. It is important to note that the Linear Programming value of a new startup node is a new upper bound for the considered allocation problem. Thus, every time a new startup node is selected, the upper bound becomes tighter if the new node's Linear Programming solution differs from that of the previous startup node. The described Branch-and-Bound search process continues until: (1) the execution time limit (5 h) is reached; (2) all the nodes in the tree are pruned except for a single leaf node, i.e., the global optimal solution is found. Typically, due to the NP-complete nature of the algorithm, stopping condition (2) is unlikely to occur when the solution space of the problem is relatively large.

An important issue for the Branch-and-Bound algorithm is the order in which applications are considered in the node expansion process. In the conducted experiments, three different orderings of applications were tested to identify the one resulting in the best performance. An arbitrary ordering implies that applications from Q strings are randomly shuffled. As opposed to such a random-based approach, a max-first ordering contains applications ranked in descending order of their average load estimate $AL^k[i]$, which is associated with each application i in string $S^k : 1 \leqslant k \leqslant Q$, and defined as

$$AL^k[i] = \frac{1}{P[k]} \times \left[\frac{1}{M} \times \sum_{j=1}^{M} \bar{u}^k[i,j] \times \bar{t}^k[i,j] + b_{\mathrm{av}} \times O^k[i] \right].$$

Applications in a min-first ordering are ranked in ascending order of their average load estimate.

6.2. Partial allocation scenario

Three major goals were addressed in the partial allocation scenario by applying the Branch-and-Bound technique based on Linear Programming formulations derived in Section 5: (1) finding a tighter bound than the initial upper bound on total worth achievable in the system for a given set of Q strings; (2) finding a tighter bound than the initial upper bound on system slackness achievable for the subset of A strings, $A \leqslant Q$; (3) making an attempt to find a better allocation for the subset of A strings to maximize system slackness. The subset of A strings, referred to in (2) and (3), is essentially the subset of mapped strings in the best chromosome produced as the Permutation Space Genitor-Based heuristic terminates, characterized by the highest known total worth value. Goals (2) and (3) with respect to the subset of A strings are identical to the goals in the complete allocation scenario with respect to the set of Q strings. As such, the Branch-and-Bound algorithm designed for the complete allocation scenario can be applied to address (2) and (3) when the set of Q strings is replaced with the subset of A strings in the corresponding Linear Programming formulation.

For goal (1), a tighter bound than the initial upper bound on total worth can be found by applying another Branch-and-Bound algorithm to find a solution for the Integer Linear Programming form given by objective function (9) and conditions (a), (b′), (c)–(f), where condition (a) is relaxed to allow for real numbers confined to the interval [0,1]. The node expansion and backtracking mechanisms in this Branch-and-Bound remain identical to those designed for the complete allocation scenario. In contrast, the tree structure considered here is different—nodes are associated with strings as opposed to the complete allocation scenario where nodes are associated with applications. Each parent node generates two children when expanded where the children represent the cases when a new string is either loaded or not loaded to the system. The term "loaded" is used here as opposed to "mapped" to emphasize that no actual application assignments are produced due to relaxed condition (a). Call a string from the set of Q strings processed when the two nodes corresponding to the loaded/not loaded decisions made for that string are included in the tree. In the Linear Programming form for a given interior node, the sum in condition (b′) is set to 0 or 1 for the processed strings, and relaxed to real numbers confined to the interval [0,1] for the others. The total worth value found by the Permutation Space

Genitor-Based heuristic is used for the initial node pruning until the Branch-and-Bound algorithm finds a better leaf solution (all strings are processed). In contrast to a large solution space in the complete allocation scenario, where the number of leaf nodes could reach $M^{\sum_{k=1}^{Q} n_k}$, the solution space that needs to be explored for the considered Integer Linear Programming form is significantly smaller, composed of at most 2^Q leaf nodes. Thus, the Branch-and-Bound heuristic is able to converge to the global optimal solution in a reasonable amount of time for a given simulation setup (about 1 h). Recall that the discussed Integer Linear Programming form does not represent the actual allocation problem due to relaxed condition (a). The order in which Q strings are processed in the tree search affects the convergence time because early improvement of the best-known leaf solution helps to curtail the search space and avoid redundant computations. Three different orderings of strings were considered in the experiments. An arbitrary ordering is based on a random arrangement of Q strings. A max-first ordering contains strings ranked in descending order of their average worth per average load estimate $AWAL[k]$, defined for each string as:

$$AWAL[k] = \frac{W[k]}{\sum_{i=1}^{n_k} AL^k[i]}.$$

Strings in a min-first ordering are ranked in ascending order of their $AWAL[k]$.

7. Simulation experiments and results

7.1. Simulation setup

The purpose of the simulation was to evaluate the performance of the developed mapping heuristics in two different workload scenarios. For each scenario, the hardware part of the intended system was composed of a heterogeneous suite of eight machines. The bandwidth of each inter-machine communication route was determined by sampling a uniform distribution on the interval between 1 and 10 Mb/s. Typically, multiple applications executing on the same machine access shared data from the physical memory system making unnecessary any physical movement of this data. Thus, all intra-machine data transmission times were set to 0, i.e., $b[j, j] = 0$. The time-of-flight, i.e., time needed for the first transmitted bit of data to reach the destination [22], was assumed to be negligible on each communication route. For all experiments, it also was assumed that an application could be executed on any machine, and its output could be transferred over any communication route. The two workload scenarios were distinguished by a different number of strings considered for mapping and different ranges for the periods of the strings.

Recall that a partial allocation scenario occurs in the oversubscribed system when not all the strings in a given set can be successfully allocated because some hardware component in the system would exceed its 100% utilization limit. To model this situation, a set of 75 strings was generated and the string periods were set to be tight, as explained below. For the complete allocation scenario 45 strings were created with more relaxed period values.

The developed heuristics were tested for operation with 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 s, and between 0.1 and 1 s, respectively. In the same fashion, the size of the output generated by each application in the string was chosen in the interval from 10 to 100 kbytes.

In addition to average nominal execution time $\bar{t}_{\text{av}}^k[i]$, introduced in Section 4, let the average time to transmit one bit of data in the system, b_{av}, be calculated as the average across all possible communication routes in the system: $b_{\text{av}} = \frac{1}{M^2}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M} b[j_1, j_2]$. The random variable μ with a uniform distribution in a particular range that was inserted to control the tightness of period $P[k]$ associated with each string. All these new variables were combined in the following way to set a period of string S^k: $P[k] = \mu \times \max\{\bar{t}_{\text{av}}^k[i] : 1 \leqslant i \leqslant n_k, b_{\text{av}} \times O^k[z] : 1 \leqslant z \leqslant n_k - 1\}$. The range for the random variable μ for the complete allocation scenario was set to [3,4.5], and for the partial allocation scenario was set to [2,3].

An interactive software framework has been developed for this study that allows for simulation, testing, and result visualization of the designed mapping techniques. The optimization package Lingo 9.0 was employed to compute Linear Programming solutions in the Branch-and-Bound algorithms. For each scenario, 50 simulation runs were performed, which allow for a 95% confidence interval [13] computation.

7.2. Experimental results

Fig. 3(a) demonstrates how the system slackness was gradually improved by the Permutation Space Genitor-Based heuristic over time for a typical run in the complete allocation scenario. In 87% of the runs the heuristic was able to generate unique offspring while performing mutation and crossover operations, and the heuristic was terminated when the second stopping criterion, i.e., 2000 iterations without a change in the best chromosome, was reached. Additional experiments revealed that an increase in the maximum number of iterations without a change in the best chromosome does not affect the performance of the heuristic. In the remaining 13% of the runs, at some point in time the Permutation Space Genitor-Based heuristic failed to produce a new unique chromosome within 10 min and was terminated. A typical run of the Permutation Space Genitor-Based heuristic lasted for less than 16 min. Fig. 3(b) shows an example where the final Permutation Space Genitor-Based heuristic's result passed to the follow-up Branch-and-Bound algorithm was improved twice. In addition to that, the upper bound on system slackness was tightened by the Branch-and-Bound as the algorithm progressed. A tight upper bound is very important to evaluate the quality of the final result—in practice such an evaluation can be used to determine when the algorithm needs to be stopped.

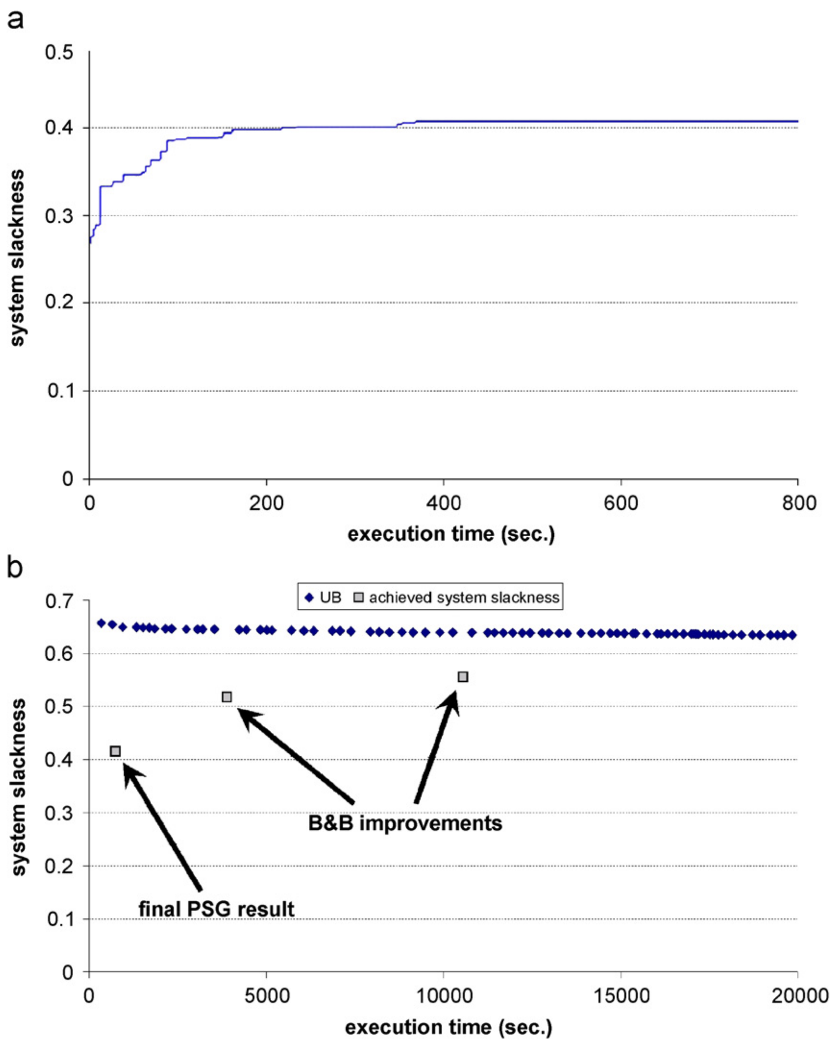


Fig. 3. System slackness in the complete allocation scenario achieved over time in a single run by applying the hybrid two-stage resource allocation method: (a) progress of the Permutation Space Genitor-Based heuristic; (b) the final result of the Permutation Space Genitor-Based heuristic passed to the follow-up Branch-and-Bound algorithm was improved twice. The upper bound on system slackness was tightened by the Branch-and-Bound as the algorithm progressed.

The performance results of 50 runs against the secondary objective metric component are shown for the complete allocation scenario in Fig. 4. In 34% of the cases, the Branch-and-Bound algorithm was able to improve the results of the Permutation Space Genitor-Based heuristic, i.e., to find a resource allocation with a higher value of system slackness. Fig. 4 depicts the best performance achieved when the max-first ordering was used to rank applications while constructing a tree in the Branch-and-Bound algorithm. In the rectangle above each bar, the top corresponds to the initial upper bound for that run and the bottom corresponds to the upper bound tightened with the Branch-and-Bound algorithm. Table 1 compares the efficiency of the max-first, min-first, and arbitrary orderings for the system slackness improvement and upper bound tightening. The results were averaged across 50 runs for each of the orderings. The highest efficiency exhibited by the max-first ordering is based on the fact that allocations for applications were resolved in the tree in descending order of their average load estimate. This ordering corresponds to the well-known bin-packing principle [29] implying that allocation is first resolved for the workload components with high resource consumption requirements. In contrast, the performance results of upper bound tightening on system slackness revealed the Branch-and-Bound algorithm to be relatively insensitive to application orderings. Averaged across 50 runs, the achieved system slackness including any Branch-and-Bound improvements was 0.47 for the complete allocation scenario, and 0.11 for the partial allocation scenario; the achieved system slackness per run normalized against the corresponding upper bound was 81% for the complete allocation scenario, and 83% for the partial allocation scenario. The much lower absolute system slackness in the partial allocation scenario must be expected: the system is "packed" with applications to the point where the remaining slack is not adequate to accommodate an extra string.

For the partial allocation scenario, the experimental results obtained with the Permutation Space Genitor-Based heuristic for the primary component of the performance metric (total worth) are shown in Fig. 5(a). The performance for each run

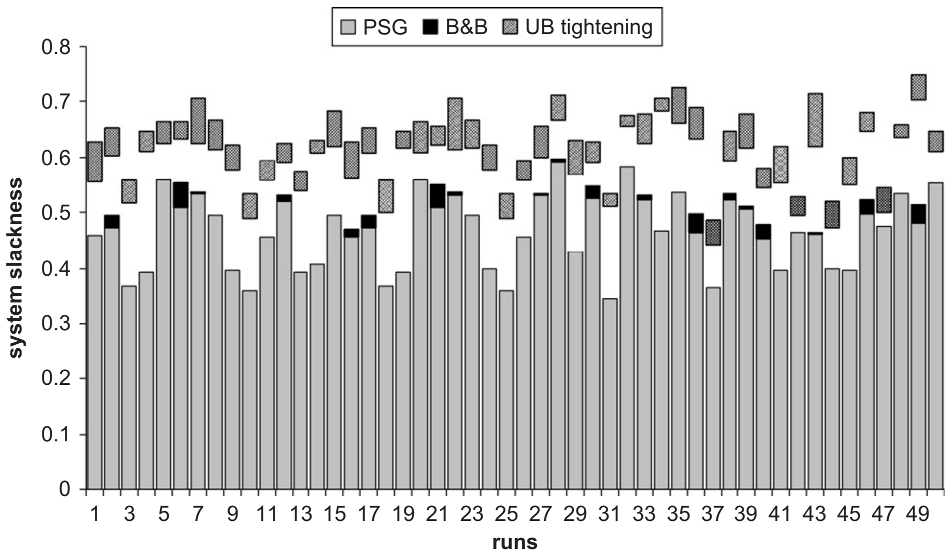


Fig. 4. The system slackness achieved by the Permutation Space Genitor-Based heuristic, and the upper bound calculation with follow-up improvements provided by the Branch-and-Bound algorithm in the complete allocation scenario for 50 experimental runs. The results are plotted against the corresponding upper bounds tightened by the Branch-and-Bound algorithm in all experimental runs. The Branch-and-Bound algorithm improved the results of the Permutation Space Genitor-Based heuristic in 34% of runs, and for these improved runs, the increment in performance averaged 4.65%. These increments are shown by the black rectangles.

Table 1
Performance of the Branch-and-Bound algorithm improving system slackness averaged across 50 runs in the complete and partial allocation scenarios

Scenorio	Ordering	System slackness improvement			Sys. slackness upper bound	
		Successful runs (%)	Improvement over first stage (%)	95% conf. interval	Average value	95% conf. interval
Complete allocation	Max-first	34	4.65	[2.92, 6.38]	0.59	[0.55, 0.63]
	Min-first	28	2.87	[0.52, 5.22]	0.59	[0.57, 0.61]
	Arbitrary	10	2.54	[1.32, 3.76]	0.59	[0.57, 0.61]
Partial allocation	Max-first	40	6.46	[4.12, 7.81]	0.32	[0.26, 0.38]
	Min-first	30	3.25	[2.36, 4.14]	0.32	[0.25, 0.39]
	Arbitrary	8	4.42	[1.78, 7.84]	0.34	[0.27, 0.41]

is plotted along with the corresponding upper bound tightened by utilizing the developed Branch-and-Bound algorithm. The convergence to a tighter upper bound was reached in 4.7 min on average across 50 runs when strings were arranged in the max-first order, 6.3 min when strings were arranged in the min-first order, and 7.4 min when strings were arranged in an arbitrary order. The Permutation Space Genitor-Based heuristic performed well in this scenario achieving mappings that averaged above 80% of the upper bound.

Fig. 5(b) illustrates the performance of the Permutation Space Genitor-Based heuristic and the follow-up max-first variant of the Branch-and-Bound heuristics maximizing the secondary component of the objective metric, i.e., system slackness. As Table 1 shows, compared to the complete allocation scenario, all three variants of the Branch-and-Bound heuristic succeeded in system slackness improvement over the results of the Permutation Space Genitor-Based heuristic in approximately the same number of runs, but their relative improvement on system slackness was higher.

As indicated before, the Branch-and-Bound heuristic requires a relatively long execution time. The effectiveness of the heuristic depends on the quality of the leaf solution used for pruning and on the size of the search space. In the successful runs, the Branch-and-Bound heuristic improved system slackness over the Permutation Space Genitor-Based heuristic in average by 4.65% (complete allocation scenario) and 6.46% (partial allocation scenario). However, when evaluated with respect to the intervals between the Permutation Space Genitor-Based heuristic's solutions and the corresponding tightened upper bounds, the average improvement is 18.73% and 24.32%, respectively. Finally, even if the Branch-and-Bound heuristic does not improve the results, it produces a tighter upper bound, so a practitioner can estimate the chance of improving the final solution and make an informed decision as to when the search process needs to be terminated.

8. Related work

A number of papers in the literature have studied the issue of finding an initial resource allocation that is robust against

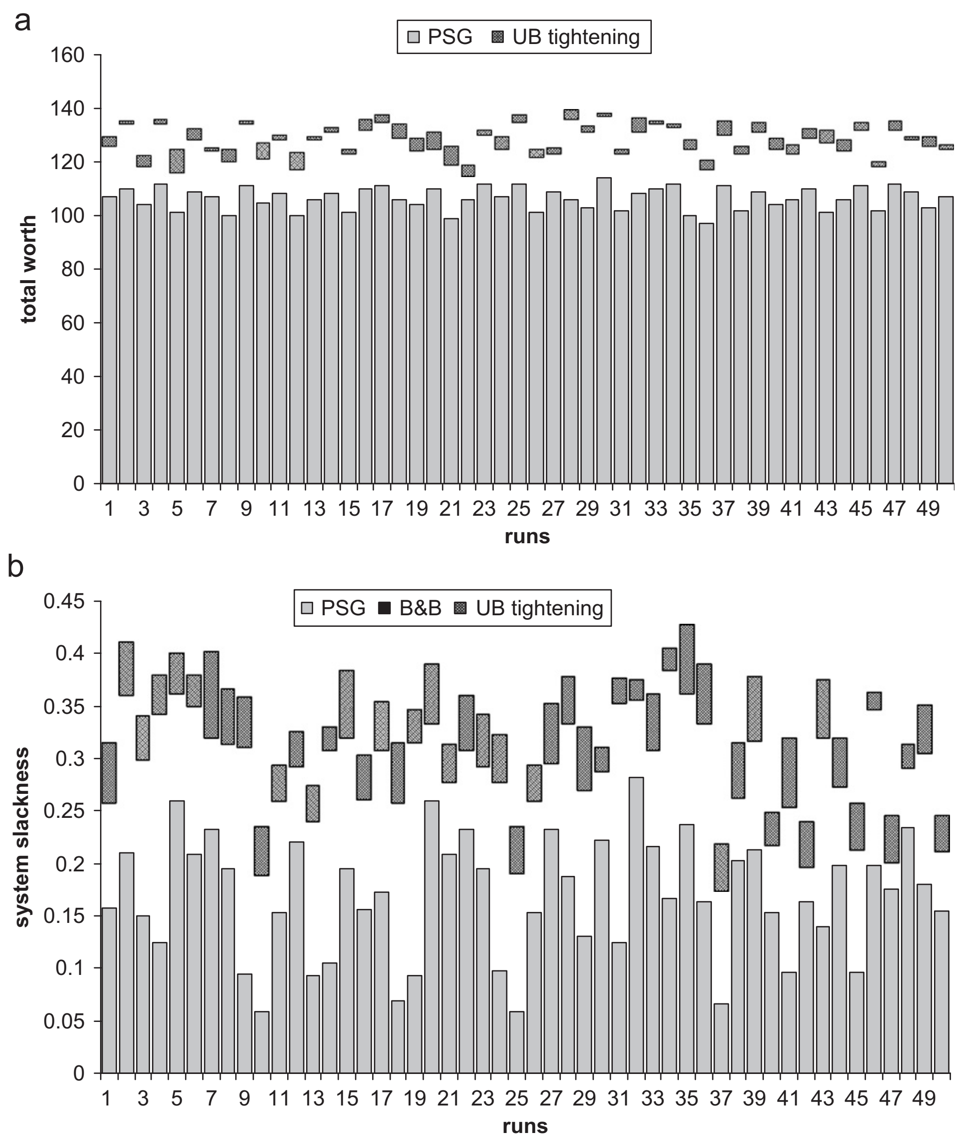


Fig. 5. Performance in the partial allocation scenario for 50 experimental runs: (a) total worth achieved by the Permutation Space Genitor-Based heuristic plotted against the upper bound on total worth tightened with the Branch-and-Bound algorithm in all experimental runs; (b) system slackness achieved by the Permutation Space Genitor-Based heuristic with a follow-up improvement provided by the Branch-and-Bound algorithm. The Branch-and-Bound improved the results of the Permutation Space Genitor-Based heuristic in 40% of runs, and for these improved runs, the increment averaged 6.46%. These increments are shown by the black rectangles. The upper bound was tightened by the Branch-and-Bound in all experimental runs.

unpredictable workload increases (e.g., [3,4,10,12,14,18,19,23]). These studies are compared below.

The nature of the problem described in [3] is similar to the presented problem in this paper. Periodically running applications are organized in sequential strings, which are subject to the imposed end-to-end latency and throughput constraints. In that study it is assumed that the computation time of an application sharing a given machine with $N - 1$ other applications was N times its nominal execution time. This results in conservative execution time estimates 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 [3].

Slack-based techniques explored in this work 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 [12,19] 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 given shipboard environment, such a function is unknown, and therefore the system slackness is an appropriate measure to use.

The research in [10] considers a single-machine scheduling environment where the processing times of individual jobs are uncertain. The system performance is measured by the total flow time (i.e., the sum of *completion* times of all jobs). Given the probabilistic information about the processing time for each job, the authors determine the normal distribution that approximates the flow time associated with a given schedule. A given schedule's robustness is then given by 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 [10], the workload increases are expected in the Adaptive and Reflective Middleware Systems 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 [10,39].

Our combination of evolutionary algorithms with the Incremental Mapping Routine is conceptually similar to [14,45]. For example, in [14] the goal is to minimize a weighted combination of the cost of the system and the execution time of a set of tasks. A genetic algorithm manipulates a set of chromosomes, where each chromosome is 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.

As opposed to heuristic scheduling algorithms finding approximate (or suboptimal) solutions, exact algorithms for finding optimal solutions are based on Integer Linear Programming. Although solving an Integer Linear Programming formulation is NP-hard, significant progress has been made in the development of efficient Integer Linear Programming algorithms. For example, Integer Linear Programming CASA schedulers for VLSI architectural synthesis, such as OASIC [18] and ALPS [23], have produced better designs than heuristic algorithms for medium-sized problems in comparable time. However, with an increase in the problem scale the performance of Integer Linear Programming schedulers degrades significantly while heuristic approaches are still able to produce high-quality solutions in a reasonable amount of time. In our work, a specially designed first-stage evolutionary heuristic was utilized to find a high-quality baseline solution that was used efficiently in the second-stage Branch-and-Bound algorithm to narrow the search. As a result, the baseline solutions were improved in at least 34% of cases for the considered resource allocation problems.

It is important to note that an implementation of both the Permutation Space Genitor-Based heuristic and Branch-and-Bound algorithm can be done in a parallel fashion. Parallel execution for Genetic Algorithms was done in [28], where the authors achieved a significant execution speed-up due to partitioning the global population into multiple subpopulations. The search for each subpopulation was then performed by a physical processing element (PPE) concurrently with other PPEs. After each iteration, the best chromosome found across all PPEs was shared and inserted into each subpopulation. The authors in [28] addressed the problem of a resource allocation for Direct Acyclic Graphs (DAGs), where the goal was to minimize makespan. In our work, the applications were organized in sequential strings (versus DAGs), and the performance metric was different from makespan. Furthermore, the throughput and latency constraints that must be satisfied in our work significantly limited the set of feasible solutions. As this set was not easily described with the operators of the Genetic Algorithm, we designed a greedy heuristic to "cast" chromosomes generated in the permutation space to feasible solutions.

The same idea of search space partitioning among multiple processing elements was presented in [25] for the A* algorithm, which mapped DAGs into a distributed computer system. Similar to Branch-and-Bound, A* is a structural exploration of the search tree, where the node with the best bound is selected for expansion. Due to NP-completeness of resource allocation problem in heterogeneous systems, both the A* and Branch-and-Bound heuristics are incapable of finding the global optimal solution(s) for a realistic size problem in a reasonable amount of time. Given the scale of the search space in our work, A* most likely will never reach a leaf solution in the allocated time and ends up with multiple partial solutions. This was our motivation for the depth-first search tree expansion method applied in Branch-and-Bound. In our design, we applied the concept of "anytime algorithms," such that the best solution found so far is always available when the algorithm is stopped. In addition, the tight upper bounds obtained with LP relaxation allowed us to prune a significant number of branches and reduce the search space. Although, the problem domains in [25,28] differ from ours, the search space partitioning and the interaction scheme among PPEs proposed in those studies can readily be adapted in our method.

9. Summary

This paper presents 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 might increase, possibly invalidating a resource allocation that was based on the initial workload estimate. 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 a measure that quantitatively reflects the system's potential to absorb unpredictable increases in input workload.

Formed by combining the efficient evolutionary Genitor-based search methods with a specially designed string allocation Incremental Mapping Routine, the Permutation Space Genitor-Based heuristic was used to generate baseline solutions in the complete and partial allocation scenarios. Further resource allocation improvement with respect to system slackness and iterative upper bound tightening for both objective metric components were based on the developed Branch-and-Bound algorithms. To establish the foundation for these algorithms, the considered resource allocation problems were formulated in the Integer Linear Programming form. Due to the high-quality of the baseline solutions provided by the Permutation Space Genitor-Based heuristic, the search spaces explored by the Branch-and-Bound algorithms were significantly reduced.

As a result, the Branch-and-Bound algorithm succeeded in 34% of the experiment runs with 4.65% improvement over the results of the Permutation Space Genitor-Based heuristic in the complete allocation scenario, and in 40% of the experiment runs with 6.46% improvement over the results in the partial allocation scenario. By demonstrating a performance ranging from 81% to 83% of the upper bound, the proposed combinatorial mapping approach indicates a significant potential to produce effective resource allocations in an environment associated with unpredictable workload increases.

Although the proposed resource allocation technique was designed for the specific Adaptive and Reflective Middleware Systems project, its application is not limited to that environment. For example, the problem of initial resource allocation for periodic applications is an important issue for embedded systems (e.g., [34,40]) and sensor networks (e.g., [48]). Similar to the environment considered in this work, many of such systems process periodic data received from various sensors producing results for actuators and must deliver a specific level of QoS while being a subject to possible workload increases. The application domains include surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems. Thus, the proposed hybrid two-stage robust resource allocation scheme can readily be adapted in those systems.

Acknowledgments

Portions of a preliminary version of this material were presented at the 14th IEEE Heterogeneous Computing Workshop (HCW 2005) [37]. The authors thank Jay Smith for his valuable comments.

Appendix A.

Table A.1 presents the glossary of notation.

Table A.1
Glossary of notation

S^k	kth string specified by a sequence of n_k applications $\{a_1^k a_2^k \ldots a_{n_k}^k\}$
$W[k]$	Worth factor of kth string
$P[k]$	Period of time between sequential raw data sets processed by kth string
$m[i,k]$	Machine to which application a_i^k is assigned
$t_{\text{comp}}^k[i]$	Estimated computation time for application a_i^k on machine $m[i,k]$
$t_{\text{tran}}^k[i]$	Estimated time to transfer output $O^k[i]$ from a_i^k to a_{i+1}^k in string S^k
$\bar{t}^k[i,j]$	Nominal data set processing time of a_i^k executing on machine j
$\bar{u}^k[i,j]$	Average CPU utilization of machine j when a_i^k processes a nominal data set
$U^{\text{machine}}[j]$	Utilization of machine j
$b[j_1,j_2]$	Time to transmit one bit of data from machine j_1 to machine j_2
$U^{\text{route}}[j_1,j_2]$	Utilization of the communication route from machine j_1 to machine j_2
M	Number of heterogeneous machines in the system
Λ	System slackness, i.e., the minimum utilization capacity remaining across all computation and communication resources
$\bar{t}_{\text{av}}^k[i]$	Average nominal execution time of a_i^k computed across M machines
$\bar{u}_{\text{av}}^k[i]$	Average nominal CPU utilization of a_i^k computed across M machines
Q	Total number of strings considered for mapping

References

[1] Adaptive and Reflective Middleware Systems (ARMS). Available online: ⟨http://dtsn.darpa.mil/ixo/ixo_FeatureDetail.asp?id = 6#⟩, 2005 (accessed 10.10.2005).

[2] S. Ali, J.-K. Kim, H.J. Siegel, A.A. Maciejewski, Static heuristics for robust resource allocation to continuously executing applications, S. Ali, personal correspondence.

[3] S. Ali, J.-K. Kim, Y. Yu, S.B. Gundala, S. Gertphol, H.J. Siegel, A.A. Maciejewski, V. Prasanna, Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system, Parallel Distributed Comput Pract. 4 (2002).

[4] S. Ali, A.A. Maciejewski, H.J. Siegel, J.-K. Kim, Measuring the robustness of a resource allocation, IEEE Trans. Parallel Distributed Systems 15 (7) (2004) 630–641.

[5] T.D. Braun, H.J. Siegel, N. Beck, L. Bölöni, R.F. Freund, D. Hensgen, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distributed Comput. 61 (6) (2001) 810–837.

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

[7] S. Chaudhuri, R.A. Walker, J.E. Mitchel, Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem, IEEE Trans. Very Large Scale Integration (VLSI) Systems 2 (4) (1994) 152–164.

[8] E.K.P. Chong, S.H. Zak, An Introduction to Optimization, second ed., Wiley, New York, NY, 2001.

[9] E.G. Coffman (Ed.), Computer and Job-Shop Scheduling Theory, Wiley, New York, NY, 1976.

[10] R.L. Daniels, J.E. Carillo, β-robust scheduling for single-machine systems with uncertain processing times, IIE Trans. 29 (11) (1997) 977–985.

[11] G.B. Dantzig, Linear Programming and Extensions, Princeton University, Princeton, NJ, 1963.

[12] A.J. Davenport, C. Gefflot, J.C. Beck, Slack-based techniques for robust schedules, in: 6th European Conference on Planning (ECP-2001), September 2001, pp. 7–18.

[13] J.L. Devore, Probability and Statistics for Engineering and Sciences, fifth ed., Duxbury Press, Los Angeles, CA, 1999.

[14] M.K. Dhodhi, I. Ahmad, R. Storer, SHEMUS: synthesis of heterogeneous multiprocessor systems, Microprocessors Microsystems 19 (6) (1995) 311–319.

[15] A. Dogan, F. Ozguner, Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems, Cluster Comput. 7 (2) (2004) 177–190.

[16] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1) (1997) 53–66.

[17] I. Foster, C. Kesselman (Eds.), The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann: San Francisco, CA, 1999.

[18] C.H. Gebotys, M.I. Elmasry, Optimal VLSI Architectural Synthesis, Kluwer Academic Publishers, New York, NY, 1992.

[19] S. Ghosh, Guaranteeing fault tolerance through scheduling in real systems, Ph.D. Dissertation, Faculty of Arts and Sciences, University of Pittsburgh, 1996.
[20] C.C. Gonzaga, Path-following methods for linear programming, SIAM Rev. 34 (2) (1992) 167–224.
[21] D. Gu, F. Drews, L.R. Welch, Robust task allocation for dynamic distributed real-time systems subject to multiple environmental parameters, in: 25th International Conference on Distributed Computing Systems (ICDCS 2005), June 2005, pp. 675–684.
[22] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, third ed., Morgan Kaufmann, San Francisco, CA, 2003 (Chapter 8).
[23] C.-T. Hwang, J.-H. Lee, Y.C. Hsu, A formal approach to the scheduling problem in high level synthesis, IEEE Trans. Comput. Aided Design Integrated Circuits Systems 10 (4) (1991) 464–475.
[24] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, J. ACM 24 (2) (1977) 280–289.
[25] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (3) (1998) 42–51.
[26] W. Keuffer, Visual Coding with Genitor, Miller Freeman, San Francisco, CA, 1997.
[27] J.-K. Kim, D.A. Hensgen, T. Kidd, H.J. Siegel, D.S. John, C. Irvine, T. Levin, N.W. Porter, V.K. Prasanna, R.F. Freund, A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems, Cluster Comput. 6 (3) (2006) 281–296.
[28] Y.-K. Kwok, I. Ahmad, Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, J. Parallel Distributed Comput. 47 (1) (1997) 58–77.
[29] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementation, Wiley, New York, NY, 1990.
[30] A.M. Mehta, J. Smith, H.J. Siegel, A.A. Maciejewski, A. Jayaseelan, B. Ye, Dynamic resoure allocation heuristics that manage tradeoff between makespan and robustness, J. Supercomput. accepted for publication (special issue on Grid Technology).
[31] Z. Michalewicz, D.B. Fogel, How to Solve it: Modern Heuristics, Springer, New York, NY, 2000.
[32] K.G. Murty, S.N. Kabadi, Some NP-complete problems in quadratic and nonlinear programming, Math. Programming 39 (2) (1987) 117–129.
[33] G. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley, New York, NY, 1999.
[34] S.I. Park, V. Raghunathan, M.B. Srivastava, Energy efficiency and fairness tradeoffs in multi-resource, multi-tasking embedded systems, in: The 2003 International Symposium on Low Power Electronics and Design (ISLPED'03), August 2003, pp. 2–13.
[35] K. Ramamritham, J. Stankovic, W. Zhao, Distributed scheduling of tasks with deadlines and resource requirements, IEEE Trans. Comput. 38 (8) (1989) 1110–1123.
[36] B. Ravindran, R.K. Devarasetty, B. Shirazi, Adaptive resource management algorithms for periodic tasks in dynamic real-time distributed systems, J. Parallel Distributed Comput. 62 (10) (2002) 1527–1547.
[37] V. Shestak, E.K.P. Chong, A.A. Maciejewski, H.J. Siegel, L. Benmohamed, I.-J. Wang, R. Daley, Resource allocation for periodic applications in a shipboard environment, 14th Heterogeneous Computing Workshop (HCW 2005), in: the Proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), April 2005, pp. 122–127.
[38] V. Shestak, J. Smith, A.A. Maciejewski, H.J. Siegel, Iterative algorithms for stochastically robust static resource allocation in periodic sensor driven clusters, in: 8th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006), November 2006, pp. 166–174.
[39] V. Shestak, J. Smith, A.A. Maciejewski, H.J. Siegel, A stochastic approach to measuring the robustness of resource allocations in distributed systems, in: International Conference on Parallel Processing (ICPP-06), August 2006, pp. 459–470.
[40] I. Shin, I. Lee, Periodic resource model for compositional real-time guarantees, 24th IEEE Real-Time System Symposium (RTSS 2003), December 2003, pp. 469–474.
[41] J. Smith, L.D. Briceño, A.A. Maciejewski, H.J. Seigel, Measuring the robustness of resource allocations in a stochastic dynamic environment, in: 21st International Parallel and Distributed Processing Symposium (IPDPS 2007), March 2007, pp. 1–10.
[42] J. Smith, V. Shestak, H.J. Siegel, S. Price, L. Teklits, P.V. Sugavanum, Resource allocation in a cluster based imaging system, in: 2007 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA 2007), June 2007, accepted for publication.
[43] P. Sugavanam, H.J. Siegel, A.A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, A. Pippin, Robust static allocation of resources for independent tasks under makespan and dollar cost constraints, J. Parallel Distributed Comput. 67 (4) (2007) 400–416.
[44] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distributed Comput. 47 (1) (1997) 8–22.
[45] J.P. Watson, L. Barbulescu, L.D. Whitley, Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance, INFORMS J. Comput. 14 (2) (2002) 98–123.
[46] D. Whitley, The genitor algorithm and selective pressure: why rank-based allocation of reproductive trials is best, in: 3rd International Conference on Genetic Algorithms, June 1989, pp. 116–121.
[47] L.A. Wolsey, Integer Programming, Wiley, New York, NY, 1998.
[48] T. Yuan, J. Boangoat, E. Ekici, F. Ozguner, Real-time task mapping and scheduling for collaborative in-network processing in dvs-enabled wireless sensor networks, in: 20st International Parallel and Distributed Processing Symposium (IPDPS 2006), April 2006, pp. 21–27.

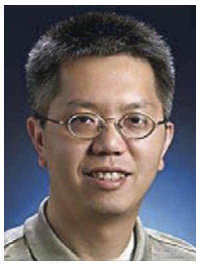
Vladimir Shestak is pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering at Colorado State University, where he has been a Research Assistant since August 2003. His current projects include resource management for clusters for IBM, Boulder. He received his M.S. degree in 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 B.S. 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. For more information, see www.engr.colostate.edu/~shestak.

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.

Howard Jay Siegel was appointed as the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in 2001, where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (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. From 1976 to 2001, he was a professor at Purdue University. Professor Siegel is a Fellow of the IEEE and a Fellow of the ACM. He received two B.S. degrees from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from Princeton University. He has coauthored over 340 technical papers. His research interests include heterogeneous parallel and distributed computing, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing and was on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He has been an international keynote speaker and tutorial lecturer and has consulted for industry and government. For more information, see www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the B.S.E.E., M.S., and Ph.D. degrees from Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently the Department Head of Electrical and Computer Engineering at Colorado State University. He is a fellow of the IEEE. A complete vita is available at: www.engr.colostate.edu/~aam.

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, NJ, 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 Principal Professional Staff with the Johns Hopkins University Applied Physics Laboratory (JHU/APL) and a Research Assistant Professor with the Computer Science Department at Johns Hopkins University. 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 post-doctoral 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 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 Laboratory. 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 17 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.