Greedy Approaches to Static Stochastic Robust Resource Allocation for Periodic Sensor Driven Distributed Systems

Vladimir Shestak*, Jay Smith†, Robert Umland*, Jennifer Hale*,
Patrick Moranville*, Anthony A. Maciejewski*, and Howard Jay Siegel*‡
*Electrical and Computer Engineering Department
‡Computer Science Department
Colorado State University, Fort Collins, CO 80523–1373
Email: {shestak, robert.umland, jennifer.hale, moranvip, hj, aam}@engr.colostate.edu
†IBM 6300 Diagonal Highway Boulder, CO 80301
Email: bigfun@us.ibm.com

***Abstract*— This research investigates the problem of robust resource allocation for a large class of systems operating on periodically updated data sets under an imposed quality of service (QoS) constraint. Such systems are expected to function in an environment replete with uncertainty where the workload is likely to fluctuate substantially. Determining a resource allocation that accounts for this uncertainty in a way that can provide a probabilistic guarantee that a given level of QoS is achieved is an important research problem. First, this paper defines a methodology for quantifiably determining a resource allocation's ability to satisfy QoS constraint in the midst of uncertainty in system parameters. Uncertainty in system parameters and its impact on system performance are modeled stochastically. Second, the established stochastic model is employed to develop greedy resource allocation heuristics. Finally, the utility of the proposed stochastic robustness metric and the performance of the heuristics are evaluated in a simulated environment that replicates a heterogeneous cluster-based radar system.**

***Index Terms*— heterogeneous distributed systems, resource allocation, stochastic optimization, greedy heuristics.**

I. Introduction

This paper investigates the problem of robust resource allocation for a large class of distributed heterogeneous computing (HC) systems operating on *periodically updated* data sets. The example systems include surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems. Fig. 1 schematically illustrates such systems where sensors (e.g., radar, sonar, video camera) produce data sets with a constant period of $\underline{\Lambda}$ time units. Due to the changing physical world, the periodic data sets produced by the system sensors typically vary in such parameters as the number of observed objects present in the radar scan and signal-to-noise ratio. Suppose that each input data set must be processed by a collection of $\underline{N}$ independent applications that can be executed in parallel in an HC system composed of $\underline{M}$ compute nodes. Unpredictable changes in input data sets result in variability in the execution times of data processing applications. This makes the problem of resource allocation (i.e., allocation of resources to applications) rather challenging, especially in situations when the system experiences workload surges or loss of hardware resources, as the total processing time for any given data set must not exceed Λ. Specifically, a produced resource allocation must be *robust*, i.e., it guarantees (or has a high probability) that the imposed timing quality of service (QoS) constraint Λ is satisfied despite uncertainties in application execution times. Furthermore, in many systems of the considered class, it is highly desirable to *minimize* the period Λ between subsequent data updates, e.g., more frequent radar scans are needed to identify an approaching target in military applications.

The major contribution of this paper is the design of resource allocation techniques based on a greedy approach methods to address the problem of minimizing Λ while providing a certain level of probabilistic guarantee that a given QoS constraint is achieved in the system. The mapping problem

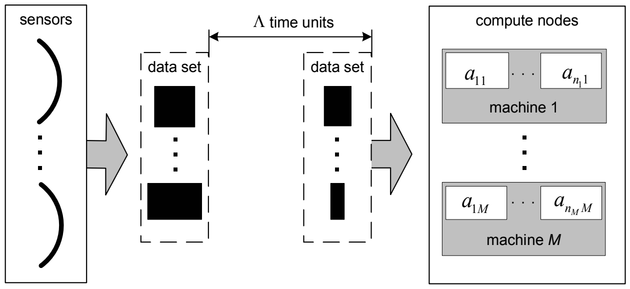

Fig. 1. Major functional units and data flow for a class of system that operates on periodically updated data sets. The a_{ij}'s denote applications executing on machine j. Processing of each data set must be completed within Λ time units.

This research was supported by the DARPA Information Exploitation Office under contract No. NBCHC030137, 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.

has been shown, in general, to be NP-complete in HC systems [6], [9], [29]. Thus, the development of heuristic techniques to find near-optimal solutions is an active area of research, e.g., [5], [10], [17], [18], [21]. Additional contributions of this research include quantifying a metric for stochastic robustness, modeling the intended distributed HC system, and evaluating the relative performance of the developed heuristics in the simulated environment.

The remainder of this work is organized in the following manner. Section II develops a model of the distributed HC system operating on periodic data sets, presents a quantitative measure of stochastic robustness of a given resource allocation along with the methods to compute it, and formulates the performance goal. Four heuristics designed to solve the resource allocation are described in Section III, each determining the lowest achievable value of the QoS constraint (i.e., Λ) for the required level of stochastic robustness. The parameters of the simulation setup are discussed in Section IV along with the simulation results and evaluation of the heuristics' performance. A sampling of some related work is presented in Section V. Section VI concludes the paper. A glossary of notation used in this paper is given in Table I.

II. Stochastic System Model

For the system illustrated in Fig. 1, let $\underline{S_j}$ be the set of $\underline{n_j}$ applications assigned to compute node $\underline{m_j}$, i.e., $S_j = \{a_{1j}, a_{2j}, ..., a_{n_j j}\}$. Let random variable $\underline{T_{ij}}$ denote the execution time of each individual application $\underline{a_{ij}}$ on compute node m_j. The random variables T_{ij} serve as the inputs to the system model that characterize the uncertainty in execution time for each of the applications in the system and will be referred to as the uncertainty parameters. The performance of the considered system is measured based on the makespan value (total time required for all applications to process a given data set) [5] achieved by a given resource allocation, i.e., a smaller makespan equates to better performance. The system performance $\underline{\psi}$ referred to as the performance characteristic, is an output of the mathematical model of the system. The functional dependence between the uncertainty parameters and the performance characteristic in the model can be expressed mathematically as

$$\psi = \max\{\sum_{i=1}^{n_1} T_{i1}, ..., \sum_{i=1}^{n_M} T_{iM}\}. \quad (1)$$

Due to its functional dependence on the uncertainty parameters T_{ij}, the performance characteristic ψ is itself a random variable.

As discussed before, the QoS constraint in the system is given by the time period Λ between sequential data sets limiting the acceptable range of possible variation in the system performance, i.e., $\psi \leq \Lambda$. **The stochastic robustness metric is the probability that the performance characteristic of the system does not exceed Λ, i.e., $\mathbb{P}[\psi \leq \Lambda]$.** For a given resource allocation, the stochastic robustness quantitatively measures the likelihood that the generated system performance will satisfy the stipulated QoS constraint. Clearly, unity is the most desirable stochastic robustness metric value, i.e., there is zero probability that the system will violate the established QoS constraint.

TABLE I
Glossary of Notation.

a_i	i^{th} application in the system
m_j	j^{th} compute note in the system
T_{ij}	execution time of a_i on m_j
Λ	time period between sequential data sets
ψ	performance characteristic
ψ_j	local performance characteristic on m_j
$\mathbb{P}[\psi \leq \Lambda]$	stochastic robustness metric
$\Lambda(a_i, m_j)$	result of PMR call when a_i is added to m_j
FFT	Fast Fourier Transform method
PMR	Period Minimization Routine
BASIC	two-phase basic heuristic
CR	two-phase conflict resolution heuristic
HI→LO	applications ranked in descending order of averages
LO→HI	applications ranked in ascending order of averages
ARBITRARY	applications ranked randomly
ITERATIVE	iterative version of a one-phase heuristic

In the model of compute node m_j, the functional dependence between the set of local uncertainty parameters $T_{ij} \mid 1 \leq i \leq n_j$ and the local performance characteristic $\underline{\psi_j}$ can be stated as $\psi_j = \sum_{i=1}^{n_j} T_{ij}$. Assuming no data dependency exists among executing applications a_{ij} on different compute nodes, the random variables $\psi_1, \psi_2, ..., \psi_M$ are mutually independent. As such, the stochastic robustness in a distributed system can be expressed as the product of the probabilities of each compute node meeting the imposed QoS constraints. Mathematically, this is given as

$$\mathbb{P}[\psi \leq \Lambda] = \prod_{j=1}^{M} \mathbb{P}[\psi_j \leq \Lambda]. \quad (2)$$

If, in addition, the execution times T_{ij} of applications mapped on a compute node m_j are mutually independent (e.g., this assumption is valid when there is no multitasking, as is commonly done in the literature [5], [8], [14], [17], [25]), then $\mathbb{P}[\psi \leq \Lambda]$ can be computed using an n_j-fold convolution of probability density functions (pdfs) $\underline{f_{T_{ij}}(t_i)}$ [15]

$$\mathbb{P}[\psi_j \leq \Lambda] = \int_0^\Lambda f_{T_{ij}}(t_1) * ... * f_{T_{n_j j}}(t_{n_j}) dt. \quad (3)$$

Often in practical implementations the probability mass functions (pmfs) $f_{T_{ij}}(t_i)$ are derived from the empirical observations obtained as a result of past executions of application a_i on compute node m_j. The discrete form of pmfs allows the Fast Fourier Transform (FFT) method [20] to be applied to efficiently compute the n-fold convolution given in (3). A more detailed discussion about the FFT method can be found in [22].

In contrast to convolution, which is applicable only when the performance characteristic is a sum of independent random variables, the bootstrap method [26] to approximate $\mathbb{P}[\psi_j \leq \Lambda]$ can be applied to *various* forms of functional dependence between local uncertainty parameters T_{ij} and the local performance characteristic ψ_j, making it very useful in many practical implementations. For example, if the execution of applications assigned to a compute node is done in a parallel multitasking fashion, then there exists a complex functional dependence between the time required to process an application and a number of currently executing threads, amount of CPU utilized by each thread, etc [21].

The bootstrap method is based on the following principles. Let T^*_{ij} denote one draw from the execution time distribution $f_{T_{ij}}(t_i)$. Let ψ^*_j be a bootstrap replication whose computation is based on a known functional dependence between the set of drawn T^*_{ij} and ψ_j, i.e., $\psi^*_j = g(T^*_{1j}, ..., T^*_{n_j j})$. In the bootstrap simulation step [26], B bootstrap replications of $\widehat{\psi}^*_j$ are computed: $\psi^*_{j,1}, ..., \psi^*_{j,B}$. If $\widehat{F}_{(B)\psi_j}(t)$ represents a sample cumulative density function (cdf) of ψ_j derived from these bootstrap replications, then the probability for the local characteristic function ψ_j can be approximated as

$$\mathbb{P}[\psi_j \le \Lambda] \approx \widehat{F}_{(B)\psi_j}(\Lambda). \tag{4}$$

The approximation in Equation (4) becomes a strict equality should the existence of a monotone normalizing transformation be assumed for the ψ_j distribution, which is based on a proof of bootstrap *percentile* confidence interval [26]. An exact normalizing transformation will rarely exist, but approximate normalizing transformations may exist—the latter causes the probability that ψ_j is less than or equal to Λ to be not exactly $\widehat{F}_{(B)\psi_j}(\Lambda)$. For additional information related to the accuracy of the bootstrap approximations refer to [22].

This research assumes that an acceptable level of stochastic robustness $\mathbb{P}[\psi \le \Lambda]$ is specified for the considered system. Thus, the performance goal for the mapper is to find resource allocations for a given set of N applications on M machines that allows for the minimum period Λ between sequential data sets while maintaining a given level of stochastic robustness.

III. Resource Allocation Techniques

A. Overview

Four greedy heuristics were designed for the problem of finding a resource allocation with respect to the performance goal stated in Section II. Greedy techniques have been adapted in many systems, e.g., [5], [10], [18], [19], as they perform well and are capable of generating solutions relatively fast as compared to time-consuming evolutionary algorithms [25], [27], [28]. The four heuristics can be categorized based on the amount of stochastic information that each of them uses. The first two of the proposed heuristics utilize the entire spectrum of stochastic information at each stage of the decision process, as opposed to the third heuristic, that uses deterministic mean values in the sorting stage, and the fourth heuristic that operates using deterministic values only. All the heuristics employ the Period Minimization Routine that is described next.

Period Minimization Routine: The PMR procedure determines the minimum possible value of Λ for a *given* resource allocation and a *given* level of stochastic robustness. As a first step, the results of n_j-fold convolutions are obtained with the FFT or bootstrap methods for each compute node corresponding to the completion time (i.e., $\sum_{i=1}^{n_j} T_{ij}$) distributions expressed in a pmf form. The completion time pmf on compute node m_j is comprised of K_j impulses, where every impulse is specified by the time $t_{kj} \mid k \in [1, K_j]$, and the probability $p_{kj} \mid k \in [1, K_j]$ for t_{kj} to occur.

As a second step, the minimum Λ is determined recursively as the smallest value among $t_{kj} \mid \{1 \le k \le K_j, 1 \le j \le M\}$, such that the specified level of stochastic robustness is less than or equal to $\prod_{j=1}^{M} \sum_{k=1}^{K_j} p_{kj} \times \mathbf{1}(t_{kj} \le \Lambda)$, where $\mathbf{1}(condition)$ is 1 if *condition* is true; 0 otherwise. The PMR procedure is summarized as follow.

```
$lo = t_1 \leftarrow \min\{t_{kj} \mid 1 \le k \le K_j, 1 \le j \le M\}$;
$hi = t_2 \leftarrow \max\{t_{kj} \mid 1 \le k \le K_j, 1 \le j \le M\}$;
$P \leftarrow$ specified level of $\mathbb{P}[\psi \le \Lambda]$;
**while** $\exists\ t_{kj} \in (lo, hi) \mid \{1 \le k \le K_j, 1 \le j \le M\}$
    $\mathbb{P}[\psi \le \Lambda] \leftarrow \prod_{j=1}^{M} \sum_{k=1}^{K_j} p_{kj} \times \mathbf{1}(t_{kj} \in [t_1, hi])$;
    **case** $\mathbb{P}[\psi \le \Lambda]$ **:**
                $== P$ : **return**;
                $> P$ : $hi \leftarrow t_2$;
                $< P$ : $lo \leftarrow t_2$;
    **end of case**
    $t_2 \leftarrow t_{kj} \mid \{1 \le k \le K_j, 1 \le j \le M\}$
        closest to $lo + (hi - lo)/2$;
**end of while**
$\Lambda \leftarrow hi$;
```

After Ω steps, the PMR procedure reduces the uncertainty range by the factor $\approx (0.5)^\Omega$, which is the fastest possible uncertainty reduction rate. This optimality becomes possible due to the fact that $\mathbb{P}[\psi \le \Lambda]$ is strictly increasing as the number of impulses considered for its computation grows. The notation $\Lambda(a_i, m_j)$ will be used to denote a PMR call that returns the minimum value of Λ for the specified level of stochastic robustness when application a_i is added to compute node m_j.

B. Two-Phase Greedy Heuristics

Basic Heuristic: The BASIC heuristic is based on the principles of the Min-Min algorithm (first presented in [10], and shown to perform well in many environments [5], [17], [18]). The heuristic traverses through N iterations resolving an allocation of one application at each iteration. In the first phase of each iteration, the heuristic determines the best assignment (according to the performance goal) for each of the applications left unmapped. In the second phase, it selects which application to map based on the best result found in the first phase. The BASIC procedure is summarized as follows.

```
**while** not all applications are mapped
    for each unmapped application $a_i$
```

```
        find the compute node $m_j$ such that
        $m_j \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid 1 \le j \le M\}$;
        resolve ties arbitrarily;
    from all $(a_i, m_j)$ pairs found above
    select the pair(s) $(a_x, m_y)$ such that
    $(a_x, m_y) \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid \text{ all } (a_i, m_j)\}$;
    resolve ties arbitrarily;
    map $a_x$ on $m_y$;
**end of while**
```

Contention Resolution Heuristic: The CR heuristic uses the sufferage concept introduced in [17]. Like the BASIC heuristic, in every iteration this heuristic first determines the best assignment for each of the applications left unmapped. Mapping decisions are finalized for those of applications whose best choice compute nodes are unique, i.e., there are no other applications competing for these nodes. In the second phase, the most critical among the competing applications gets allocated, determined as the application with the largest difference between the minimum Λ values corresponding to this application assignments to its best choice and the second best choice compute nodes. The CR procedure is summarized as follows.

```
**while** not all applications are mapped
    for each unmapped application $a_i$
        find the first choice compute node $m_j^{(1)}$ as
        $m_j^{(1)} \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid 1 \le j \le M\}$;
    for all $(a_i, m_j^{(1)})$ pairs found above
        if $m_j^{(1)}$ is unique then map $a_i$ on $m_j^{(1)}$;
        else find the second choice compute node $m_j^{(2)}$ as
            $m_j^{(2)} \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid (1 \le j \le M$
                                         $\& \; m_j \ne m_j^{(1)})\}$;
            compute contention value $C(a_i)$ as
            $C(a_i) \leftarrow \{\Lambda(a_i, m_j^{(2)}) - \Lambda(a_i, m_j^{(1)})\}$;
    select unmapped application with maximum $C(a_i)$
    $a_x \leftarrow \text{argmax}\{C(a_i) \mid \text{all unmapped } a_i\}$;
    resolve ties arbitrarily;
    map $a_x$ on its first choice compute node;
**end of while**
```

C. One-Phase Greedy Heuristics

Sorting Heuristic: This heuristic inherits the concepts developed for the MCT algorithm that was observed to perform well adopted in multiple resource allocation schemes designed for distributed systems [5], [17]. Initially, all N applications considered for mapping are sorted based on the average computed for each application across the mean values $\mu(T_{ij})$ derived from execution time distributions of $T_{ij} \mid 1 \le j \le M$. Three different orderings were considered in the experiments. A HI→LO ordering contains applications ranked in descending order of their averages; applications in LO→HI ordering are ranked in ascending order of their averages; and ARBITRARY ordering implies that N applications are randomly shuffled. Once sorting is completed, applications are fetched sequentially, each mapped on the compute node selected to provide the minimum value of period Λ under the imposed level of stochastic robustness. The heuristic's procedure is summarized as follows.

```
for each application $a_i$
    find the average $A(a_i)$ as
    $A(a_i) \leftarrow \left[\sum_{j=1}^{M} \mu(T_{ij})\right] / M$;
order applications based on their averages $A(a_i)$
according to the selected type {HI→LO, LO→HI,
ARBITRARY};
**while** not all applications are mapped
    fetch unmapped application $a_i$ from the sorted list;
    find the compute node $m_j$ such that
    $m_j \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid 1 \le j \le M\}$;
    resolve ties arbitrarily;
    map $a_i$ on $m_j$;
**end of while**
```

Mean Load Balancing Heuristic: This heuristic was developed based on the concepts of the OLB algorithm discussed in [13], [17]. First, the N applications are sorted based on average value, as in the sorting heuristic. Then, the applications are mapped in the {HI→LO, LO→HI, ARBITRARY} order where the compute node with the minimum mean of its execution time distribution is selected for each allocation. The heuristic's procedure is summarized as follows.

```
for each application $a_i$
    find the average $A(a_i)$ as
    $A(a_i) \leftarrow \left[\sum_{j=1}^{M} \mu(T_{ij})\right] / M$;
order applications based on their averages $A(a_i)$
according to the selected type {HI→LO, LO→HI,
ARBITRARY};
**while** not all applications are mapped
    fetch unmapped application $a_i$ from the sorted list;
    find the compute node $m_j$ such that
    $m_j \leftarrow \text{argmin}\{\mu(T_{ij}) \mid 1 \le j \le M\}$;
    resolve ties arbitrarily;
    map $a_i$ on $m_j$;
**end of while**
```

IV. Simulation Experiments and Results

To evaluate the performance of the greedy heuristics described in Section III for the considered class of distributed HC systems operating on periodic data, the following approach was used to simulate a cluster-based radar system environment schematically illustrated in Fig. 1. The execution time distributions for twenty eight different types of possible radar ray processing algorithms on eight ($M = 8$) heterogeneous compute nodes were generated by combining experimental data and benchmark results. The experimental data, represented by two execution time sample pmfs, were obtained by conducting experiments on the Colorado MA1 radar [12]. These sample pmfs contain times taken to process 500 radar rays of different complexity by the Pulse-Pair & Attenuation Correction algorithm [3] and by the Random Phase & Attenuation Correction algorithm [3], both executed in non-multitasking mode on the Sun Microsystems Sun Fire V20z workstation. To simulate the effect of executing these algorithms on different platforms, each sample pmf was scaled by a factor corresponding to the performance ratio of a Sun Microsystems Sun Fire V20z to each of eight selected compute nodes[1] based on the results of the fourteen floating point benchmarks from the CFP2000 suite [23]. Combining the results available from CFP2000 for fourteen different benchmarks on eight selected compute nodes and two sample pmfs provided a means for generating a 28×8 matrix where the ij^{th} element corresponds to the execution time distribution of a possible ray processing algorithm of type i on compute node j.

A set of 128 applications ($N = 128$) was formed for each of 50 simulation trials, where for each trial the type of each application was determined by randomly sampling integers in the range $[1, 28]$. The 50 simulation trials provide good estimates of the mean and 95% confidence interval computed for every heuristic.

The results of the simulation are presented in Fig. 2. Both two-phase heuristics perform comparably and significantly outperform the other heuristics. By utilizing the entire spectrum of stochastic information at each stage of the decision process these two heuristics are able to outperform the others, in terms of minimizing Λ.

All of the variants of the one-phase sorting heuristic (the results for ARBITRARY ordering represent the average obtained over 50 reshuffled application orderings) performed consistently better than the mean load balancing heuristic variants but worse than two-phase heuristics. Recall that the sorting algorithm utilizes all of the available stochastic information to select individual task machine pairings but relies on deterministic information to order tasks for their selection. By utilizing a task ordering process that relies on deterministic information only, the number of required convolutions to produce a mapping is drastically reduced but the quality of the mapping is also affected. For example, the first two-phase heuristic required approximately 66,000 1-fold convolutions to produce a mapping, whereas the one-phase sorting heuristic required only 1024 1-fold convolutions to construct a mapping. This difference in the number of convolutions directly translated into a roughly 30 times reduction in the execution time of a simulation trial using the latter heuristic.

Finally, the one-phase mean load balancing heuristic consistently performed the worst because it ignores the available stochastic information about task execution times. This results in ignoring the impact of machine heterogeneity on the completion time distributions, which is reflected in a high Λ value. Because the one-phase mean load balancing heuristic only operates with the means of execution time distributions during the mapping process, this heuristic avoided time-consuming convolution calls. This enabled OPMLB to finish in a small fraction of the time required for either two-phase heuristic to generate a mapping.

Once the simulation results had been collected for the developed heuristics, it was noticed that there was a large discrepancy in the amount of computation required to produce each of the various mappings, i.e., two-phase heuristics required tens of thousands of convolutions to produce a mapping as opposed to one-phase techniques required 1024 or less. Consequently, two new variants of the one-phase algorithms that use multiple iterations, denoted in Fig. 2 as ITERATIVE, were created to increase the number of evaluated solutions to the level of BASIC and CR, i.e., enable these variants to utilize roughly the same amount of computation to produce a mapping.

In both iterative variants, a random restart step was introduced so that after a mapping is produced a new random ordering is generated and the heuristic is executed again. Upon completion of each iteration the resultant mapping is compared against the best mapping found so far by previous iterations. If the new mapping is an improvement on the best mapping, then it is retained as the new best mapping, otherwise it is discarded.

The results of the iterative variants are plotted in Fig. 2. As can be expected, the results of both iterative approaches demonstrated some improvement over their non-iterative versions. However, the iterative version of the sorting heuristic performed worse than the BASIC heuristic (the confidence intervals of the two do not overlap) but is a marked improvement over the corresponding non-iterative version. The average Λ over 50 trials of the BASIC heuristic was 542.5 msec.; whereas the average Λ over 50 trials of the iterative version of the sorting heuristic was 569.7 msec.—each had a confidence interval of 7 msec. The performance demonstrated by the iterative version of the mean load balancing heuristic was still significantly worse than the performance of the other heuristics.

V. Related Work

Often, modern parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of QoS. This reality inspired an increasing interest in robust resource allocation design in such systems and called for a foundation of a universal robustness framework.

The latter issue was first addressed in [1], where the authors proposed a four-step FePIA procedure to derive a robustness metric for a given resource allocation in a distributed system. The framework was based on *deterministic* estimates (e.g., current or nominal values) for each of the considered system parameters. As a measure of robustness, the "minimum robustness radius" was used that indicates the

[1]The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

distance from the current (or nominal) state of the system in a multidimensional space of perturbation parameters to the nearest point where a QoS violation occurs. Assuming no a priori information available about the relative likelihood or magnitude of change for each perturbation parameter, the minimum robustness radius implies a deterministic worst-case analysis. In our stochastic model, more information regarding the variation in the perturbation parameters is assumed known. Representing the uncertainty parameters of the system as stochastic variables enables the robustness metric in the stochastic model to account for all possible outcomes for the performance of the system. An example comparison analysis between the stochastic robustness metric and deterministic one is given in [22]. The stochastic robustness metric requires more information and, in general, is far more complex to calculate than its deterministic counterpart.

In [4], the robustness of a resource allocation is defined in terms of the schedule's ability to tolerate an increase in application execution time without increasing the total execution time of the resource allocation. A resource allocation's robustness implies system slack thereby the authors are focusing their metric on a single very important uncertainty parameter, i.e., variations in application execution times. Our metric is more generally applicable, allowing for any definition of QoS and able to incorporate any identified uncertainty parameters.

Our methodology relies heavily on an ability to model the uncertainty parameters as stochastic variables. Several previous efforts have established a variety of techniques for modeling the stochastic behavior of application execution times [2], [7], [16]. In [2], three methods for obtaining probability distributions for task execution times are presented. The authors also present a means for combining stochastic task representations to determine task completion time distributions. Our work leverages this method of combining independent task execution time distributions and extends it by defining a means for measuring the robustness of a resource allocation against an expressed set of QoS constraints.

In [11], a procedure for predicting task execution times is presented. The authors introduce a methodology for defining data driven estimates in a heterogeneous computing environment based on nonparametric inference. The proposed method is applied to the problem of generating an application execution time prediction given a set of observations of that application's past execution times on different compute nodes. The model defines an application execution time random variable as the combination of two elements. The first element corresponds to a vector of known factors that have an impact on the execution time of the application and is considered to be a mean of the execution time random variable. A second element accounts for all unmodeled factors that may impact the execution time of an application and is used to compute a sample variance. Potentially, this method can be extended to determine probability density functions describing the input random variables in our framework.

The deterministic robustness metric established for distributed systems in [1] was used in multiple heuristics approaches presented in [24]. Two variations of robust mapping of independent tasks to machines were studied in that research. In the fixed machine suite variation, six static heuristics were presented that maximize the robustness of a mapping against aggregate errors in the execution time estimates. The variety of evolutionary algorithms, e.g., Genitor and Memetic Algorithm, demonstrate higher performance as compared to the non-iterative greedy heuristics. However, greedy heuristics required significantly less time to complete a mapping. A similar trade-off was observed for another variation where a set of machines must be selected under a given dollar cost constraint that will maximize the robustness of a mapping. In our study, greedy heuristics applied in a stochastic domain experienced a significant execution slowdown due to a substantial number of calls for a convolution routine required at each step of a mapping "construction" process. Although this indicate that greedy heuristics may be inappropriate choices for systems

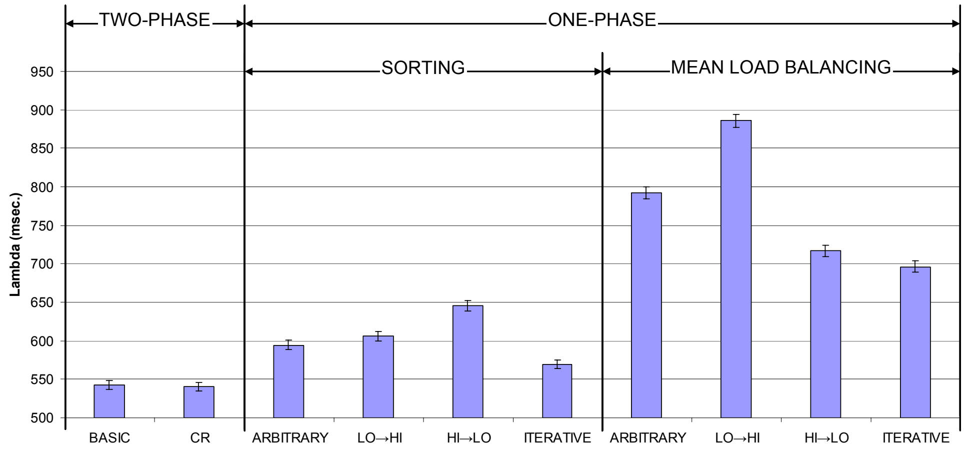

Fig. 2. A comparison of the results obtained for the described heuristics where the minimum acceptable robustness value was set to be 0.90. The y-axis corresponds to a Λ value obtained by executing the corresponding heuristics. The Λ value for each heuristic corresponds to the average over 50 trials.

where the time allotted to produce a mapping is strictly limited, the application of a bootstrap approximation method presented in Section II can alleviate such a problem in some of those systems.

In [8], the authors present a derivation of the makespan problem that relies on a stochastic representation of task execution times. The paper demonstrates that the proposed stochastic approach to scheduling can significantly reduce the actual simulated system makespan as compared to some well known scheduling heuristics that are founded in a deterministic approach to modeling task execution times. The heuristics presented in that study were developed to minimize the expected system makespan given a stochastic model of task execution times. In contrast to [8] in our research, the emphasis was to build resource allocations capable of maintaining a certain level of probability to deliver on expressed QoS constraints by applying greedy allocation techniques.

VI. Conclusion

This paper presents a set of greedy approaches for resource allocation based on our stochastic robustness metric. The stochastic robustness metric is suitable for evaluating the likelihood that a resource allocation will perform acceptably, i.e., satisfy imposed QoS constraints, despite an uncertainty in system parameters. The stochastic robustness metric was applied in a simulated environment to an example class of systems operating with periodic data sets to demonstrate its utility in evaluating the robustness of a resource allocation.

An extensive simulation study was conducted to evaluate the performance of multiple greedy approaches to the considered resource allocation problem. The goal of the simulation was to minimize the required computation time to process periodic sensor generated data. The two-phase heuristics demonstrated great potential for practical implementations in this environment.

References

[1] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630–641, July 2004.

[2] G. Bernat, A. Colin, and S. M. Peters, "WCET analysis of probabilistic hard real-time systems," in *Proceedings 23rd IEEE Real-Time Systems Symposium (RTSS '02)*, 2002.

[3] N. Bharadwaj and V. Chandrasekar, "Waveform design for CASA X-band radars," in *Proceedings 32st Conference on Radar Meteorology of American Meteorology Society*, Oct. 2005.

[4] L. Bölöni and D. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, pp. 395–412, Sept. 2002.

[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, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, June 2001.

[6] E. G. Coffman, Ed., *Computer and Job-Shop Scheduling Theory*. New York, NY: John Wiley & Sons, 1976.

[7] L. David and I. Puaut, "Static determination of probabilistic execution times," in *Proceedings 16th Euromicro Conference on Real-Time Systems (ECRTS '04)*, June 2004.

[8] A. Dogan and F. Ozguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

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

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

[11] M. A. Iverson, F. Ozguner, and L. Potter, "Statistical prediction of task execution times through analytical benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1374–1379, Dec. 1999.

[12] F. Junyent, V. Chandrasekar, D. McLaughlin, S. Frasier, E. Insanic, R. Ahmed, N. Bharadwaj, E. Knapp, L. Krnan, and R. Tessier, "Salient features of radar nodes of the first generation NetRad system," in *Proceedings IEEE International Geoscience and Remote Sensing Symposium 2005 (IGARSS '05)*, July 2005, pp. 420–423.

[13] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic mapping in energy constrained heterogeneous computing systems," in *Proceedings 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Apr. 2005.

[14] A. Kumar and R. Shorey, "Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, Oct. 1993.

[15] A. Leon-Garcia, *Probability & Random Processes for Electrical Engineering*. Reading, MA: Addison Wesley, 1989.

[16] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 35–52, July 1997.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–121, Nov. 1999.

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

[19] G. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York, NY: John Wiley & Sons, 1999.

[20] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems, and Transforms*. Upper Saddle River, NJ: Pearson Education, 2003.

[21] V. Shestak, E. K. P. Chong, A. A. Maciejewski, H. J. Siegel, L. Benmohamed, I.-J. Wang, and R. Daley, "Resource allocation for periodic applications in a shipboard environment," in *Proceedings 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), 14th Heterogeneous Computing Workshop (HCW 2005)*, Apr. 2005, pp. 122–127.

[22] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "A stochastic approach to measuring the robustness of resource allocations in distributed systems," in *International Conference on Parallel Processing (ICPP–06)*, 2006, accepted, to appear.

[23] Standard performance evaluation corporation. Accessed Feb. 6, 2006. [Online]. Available: http://www.spec.org/

[24] 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, and A. Pippin, "Robust static allocation of resources for independent tasks under makespan and dollar cost constraints," *Journal of Parallel and Distributed Computing*, 2006, accepted to appear.

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

[26] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*. New York, NY: Springer Science+Business Media, 2005.

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

[28] D. Whitley, "The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *Proceedings 3rd International Conference on Genetic Algorithms*, June 1989, pp. 116–121.

[29] L. A. Wolsey, *Integer Programming*. New York, NY: John Wiley & Sons, 1998.