Proceedings of the 18th IASTED International Conference
Parallel and Distributed Computing and Systems
November 13–15, 2006, Dallas, TX, USA
ISBN Hardcopy: 0-88986-638-4 / CD: 0-88986-640-6

ITERATIVE ALGORITHMS FOR STOCHASTICALLY ROBUST STATIC RESOURCE ALLOCATION IN PERIODIC SENSOR DRIVEN CLUSTERS

Vladimir Shestak[1,3], Jay Smith[3], Anthony A. Maciejewski[1], and Howard Jay Siegel [1,2]

[1]Electrical and Computer Engineering Department
[2]Computer Science Department,
Colorado State University, Fort Collins, CO 80523–1373,
Email:{shestak, hj, aam}@engr.colostate.edu
[3]IBM, 6300 Diagonal Highway Boulder, CO 80301
Email: {vshestak, bigfun }@us.ibm.com

ABSTRACT
This research investigates the problem of robust static resource allocation for a large class of clusters processing periodically updated data sets under an imposed quality of service constraint. The target hardware platform consists of a number of sensors generating input for heterogeneous applications continuously executing on a set of heterogeneous compute nodes. In practice such systems are expected to function in a physical environment replete with uncertainty, which causes the amount of processing required over time 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. The stochastic robustness metric is based on a mathematical model where the relationship between uncertainty in system parameters and its impact on system performance is described stochastically. The established metric is then used in the design of several resource allocation algorithms utilizing evolutionary approaches. The performance results and comparison analysis are presented for a simulated environment that replicates a heterogeneous cluster-based processing center for a radar system.

KEY WORDS
Heterogeneous systems, resource allocation, stochastic optimization, iterative algorithms, computer clusters.

1 Introduction

This paper investigates the problem of robust resource allocation for a large class of heterogeneous cluster (HC) systems operating on *periodically updated* data sets. The application domains 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 Λ time units. Due to the changing physical world, these periodic data sets 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 N independent applications that can be executed in parallel in an HC system composed of 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 a specified *percentage* of data sets must not exceed Λ. *Robust* design for such systems involves determining a resource allocation that can account for uncertainty in application execution times in a way that enables a probabilistic guarantee for a given percentage. 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.

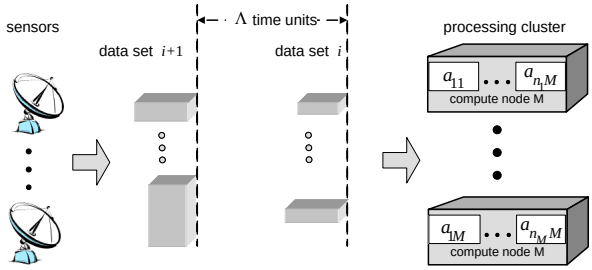

Figure 1. Major functional units and data flow for a class of computer clusters that operates on periodically updated data sets.

The major contribution of this paper is the design of resource allocation techniques based on *iterative* algorithms [20] to address the problem of minimizing Λ while providing a certain level of probabilistic guarantee that the time required for a total data set processing is less than or equal to Λ. The resource allocation problem has been shown, in general, to be NP-complete in HC systems [6, 10, 31]. Thus, the development of heuristic techniques to find near-optimal solutions is an active area of research, e.g., [5, 13, 19, 20, 23]. The notion of stochastic robustness, considered as a constraint in the addressed optimization problem, was established in [24] based on a de-

*This research was supported by NSF under grant No. CNS-0615170, 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.

veloped mathematical model of this class of HC systems. Three *greedy* approaches to the resource allocation problem were then designed presented in [25]. The latter study revealed that the adopted greedy techniques turned out to be rather time-consuming while operating in the stochastic domain. As the research continued, more sophisticated iterative algorithms were created resulting in a significant performance improvement. These iterative techniques are in focus in this paper.

The remainder of this work is organized in the following manner. Section 2 develops a model of the HC system operating on periodic data sets, presents a quantitative measure of the stochastic robustness of a given resource allocation along with methods to compute it, and formulates the performance goal. Three iterative algorithms designed to solve the resource allocation are described in Section 4, each determining the lowest achievable value of period Λ for the required level of stochastic robustness. The parameters of the simulation setup are discussed in Section 6 along with the simulation results and evaluation of the heuristics' performance. A sampling of some related work is presented in Section 7. Section 8 concludes the paper.

2 Stochastic Robustness Framework

2.1 Definition of Stochastic Robustness

The derivation of a stochastic robustness metric for a given HC environment requires a mathematical model that accounts for uncertainty in system parameters to reasonably predict the performance of the system. To emphasize the distinction between the system and its mathematical model, any new terminology related to the model will explicitly reference the model.

For the system illustrated in Fig. 1, let n_j applications assigned to compute node η_j. Let random variable T_{ij} denote the execution time of each individual application a_{ij} on compute node η_j. The random variables T_{ij} serve as the inputs to the mathematical 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. This work assumes that the distribution is known for each of the uncertainty parameters described with a probability density function (pdf). The above assumption can be justified by the fact that the majority of production HC systems execute a certain set of data processing applications for a substantial amount of time. This makes it possible to accumulate enough statistics capturing the behaviour of uncertainty parameters. Widely used methods to derive pdfs from statistical data can be found in [29]. 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 ψ 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, once system is put into operation, the time period Λ between sequential data sets is fixed limiting the acceptable range of possible variation in the system performance, i.e., $\psi \leq \Lambda$. **The stochastic robustness metric, denoted as θ, is the probability that the performance characteristic of the system does not exceed Λ, i.e., $\theta = \mathbb{P}[\psi \leq \Lambda]$.** For a given resource allocation, the stochastic robustness quantitatively measures the likelihood that the total time required to process a data set will not exceed the period between sequential data updates. 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.

In the model of compute node η_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 ψ_j can be stated as $\psi_j = \sum_{i=1}^{n_j} T_{ij}$. Assuming no inter-application data transfers exist among the applications a_{ij} executing on different compute nodes, random variables $\psi_1, \psi_2, ..., \psi_M$ are mutually independent. As such, the stochastic robustness in HC system can be found as a product of the probabilities that execution on each compute node satisfies the imposed Λ level. Mathematically, this is given as

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

If execution times T_{ij} of applications mapped on a compute node η_j are mutually independent—i.e., there are no inter-application data transfers and no multitasking; the latter simplifying assumption is commonly made in the literature [5, 8, 16, 19, 28]—then each multiplication term in (2) can be computed using an $(n_j - 1)$-fold convolution of probability density functions $f_{T_{ij}}(t_i)$ [17]

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

This research assumes that an acceptable level of stochastic robustness 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 compute nodes that allows for the minimum period Λ between sequential data sets while maintaining a given level of stochastic robustness θ.

2.2 Computational Methods Applied

Often in practical implementations the *sample* probability mass functions (pmfs) $f_{T_{ij}}(t_i)$ are derived from empirical observations obtained as a result of past executions of application a_i on compute node η_j [29]. The discrete form allows the Fast Fourier Transform (FFT) method [22] to be used to efficiently compute the convolution integral given in (3). A more detailed discussion about nuances related to the FFT method applied in the given system can be found in [24].

In contrast to convolution, which is applicable only when the performance characteristic is a sum of independent random variables, the bootstrap method [29] 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 [23], etc.

The bootstrap approximation scheme adapted for HC environment is described in [24]. A brief overview of the principles the bootstrap method is based on are as follows. 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 [29], 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

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

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 [29]. 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 [24].

3 Simulation Setup

To evaluate the performance of the iterative heuristics described in Section 4 for the considered class of 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 [15]. 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 [26]. 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 yj^{th} element corresponds to the execution time distribution of a possible ray processing algorithm of type y on compute node η_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 resource allocation algorithm.

[1]The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER 650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

4 Iterative Resource Allocation Techniques

4.1 Overview

Three iterative algorithms were designed for the problem of finding a resource allocation with respect to the performance goal stated in Subsection 2.1. Iterative algorithms are probabilistic search techniques that have been widely used as a tool in optimization [20, 23, 30], artificial intelligence [12], and many other areas. The first two of the developed heuristics operate with a set of complete resource allocations; whereas the third heuristic iteratively changes a single complete resource allocation. As opposed to greedy algorithms investigated in our previous work, where a single complete resource allocation was "constructed" [25], iterative heuristics progress toward a final solution through modified versions of complete resource allocations. In each iteration, the existing complete resource allocation (or set of allocations) is modified and evaluated. Such an iterative search process continues until an appropriate stopping criterion is reached.

To establish a basis for the comparison of the developed iterative algorithms and demonstrate the performance over time for each of them, a common stopping criterion (CSC) of 150,000 evaluations of the produced resource allocations was used in this study. It is important to note that the evaluation in the considered stochastic domain is the most computationally intensive part of any of the developed algorithms as it calls for M executions of (n_j-1)-fold convolutions, followed by a recursive search for a minimum Λ level. The evaluation mechanism, referred to as the Period Minimization Routine, 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 $(n_j - 1)$-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 for compute node η_j is comprised of K_j impulses, where every impulse corresponds to a possible pair of time outcome t_{kj} and associated probability p_{kj} for $k \in [1, K_j]$.

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

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

Figure 2. The Period Minimization Routine procedure.

The PMR procedure is summarized in Fig. 2.

After Ω iterations, 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 θ is strictly increasing as the number of impulses considered for its computation grows.

4.2 Steady State Genetic Algorithm

The adopted genetic algorithm (GA) implementation (summarized in Fig. 3) was motivated by the Genitor evolutionary heuristic [30]. Each chromosome in the GA models a complete resource allocation as a vector of numbers of length N where the i^{th} element of the vector identifies the compute node assignment for application a_i. The order in which applications are placed in a chromosome does not play any role and can be considered arbitrary. The population size for the GA was fixed at 200 for each iteration. The population size was chosen experimentally by varying the population size between 100 and 250 in increments of 50. For the samples tried, a value of 200 performed the best and was chosen for all trials. The initial members of the population were generated by applying the one-phase sorting greedy heuristic presented in [25], in which the application ordering was perturbed to produce different resource allocations to serve as the initial members of the population. In addition, the solution produced by the BASIC heuristic from [25] was also added to the initial population.

The GA used in this work was implemented as a *steady state* GA, i.e., for each iteration of the GA only a single pair of chromosomes will be selected for crossover. Selection for crossover was implemented as rank-based selection using linear bias function [30] where the population of chromosomes is sorted according to evaluation of Λ values. The most fit chromosome corresponds to a resource allocation with the smallest Λ value supportable at the specified level of stochastic robustness θ. Each chromosome generated by crossover or mutation is inserted into the population according to its evaluation such that after insertion the population remains sorted. After insertion the population is truncated to the original population size.

To maintain the selective pressure of rank-based selection an additional constraint was placed on the population where each chromosome must be unique, i.e., clones are explicitly disallowed. If a chromosome produced in any iteration were to generate a clone of an individual already present in the population or the graveyard, then that clone would be discarded prior to its evaluation for insertion into the population.

To reduce the number of duplicate chromosome evaluations, each chromosome that is trimmed from the active population is recorded in a list of known bad chromosomes referred to as the graveyard. Selecting the size of the graveyard reflected a trade-off between the time required to identify that a new chromosome was not present in the population and graveyard and the time required to evaluate the new chromosome. The graveyard size was limited to 20,000 chromosomes.

The crossover operator was implemented using a two-point reduced surrogate procedure [30] where the elements between the crossover points are exchanged between the two parents. Crossover points are selected such that at least one element of the parent chromosomes differs between the selected crossover points as this guarantees offspring to not be a clone of its parents. In addition, each generated offspring is checked for uniqueness in the population and graveyard prior to Λ evaluation.

The final step in a single iteration of the GA is mutation. For each iteration of the GA, the mutation operator is applied to the newly generated offspring of the crossover operator. Each application assignment of the offspring is individually mutated with a probability referred to as the mutation rate. For the simulated environment, the best results were achieved using a mutation rate of 0.01. For a given application, the mutation operator randomly selects a different compute node assignment from a subset of compute nodes. The subset includes compute nodes providing the smallest mean execution times for the given application. The best results in the simulation study were achieved when the size of this subset was set to three. Following mutation a final local search procedure, conceptually analogous to steepest descent technique, was applied to the result prior to inserting the mutated chromosome into the population.

A local search operator was introduced for inclusion into a GA as a follow on step to the mutation operator for a flowshop problem in [32]. The implementation of the local search procedure, follows that of the coarse refinement presented as part of the GIM heuristic described in [27]. In particular, all applications are examined to determine which of them should be moved to a different compute node to realize the largest decrease in the minimum supportable Λ value. The procedure continues until moving any application would result in an increase in the minimum supportable Λ value.

4.3 Ant Colony Optimization

The Ant Colony Optimization (ACO) heuristic (summarized in Fig. 4) belongs to a class of swarm optimization algorithms where low-level interactions between artificial (i.e., simulated) ants result in large-scale optimizations by the larger ant colony. The technique was inspired by colonies of real ants that deposit a chemical substance (pheromone) when searching for food. This substance influences the behavior of individual ants. The greater the amount of pheromone on a particular path, the larger the probability that an ant will select that path. Artificial ants in ACO behave in a similar manner by recording their chosen path in a global pheromone table.

```
generate initial population;
evaluate each chromosome;
rank population based on Λ values;
while CSC not met
    select two chromosomes from the population;
    select crossover points;
    exchange compute node assignments
    between crossover points;
    ascertain if either offspring are unique;
    for each element of each child chromosome
        generate a random number x in the range [0,1];
        if x < mutation rate
            determine 3 minimum mean execution time machines
            for the selected application;
            arbitrarily change the compute node assignment
            of the selected application;
    apply local search to each of the offspring;
    ascertain if either offspring is unique;
    insert unique offspring into population;
    trim population down to population size;
    move dead chromosomes to the graveyard;
end of while
output the best solution.
```

Figure 3. The steady state Genetic Algorithm procedure.

The ACO algorithm implemented here is a variation of the ACO algorithm design described in [9]. During ACO execution, the $N \times M$ pheromone table is maintained and updated allowing the ants to share global information about good compute nodes for each application. Let each element of pheromone table, denoted as $\tau(a_i, \eta_j)$, represent the "goodness" of compute node η_j for application a_i. At a high level, the ACO heuristic works in the following way. A certain number of ants are released to find different complete mapping solutions. Based on the mapping produced by the individual ants, the pheromone table is updated. This procedure is repeated as long as the common stopping criterion is not reached. The final mapping solution is determined by mapping each application to its highest pheromone value compute node.

At a low level, each ant heuristically "constructs" its complete mapping, and its mapping decision process balances between the (a) performance metric and (b) pheromone table information. The ant procedure involves two phases. In Phase 1, for each unmapped application, the compute node, denoted as $m_{best}(a_i)$, is determined such that it would provide the minimum mean completion time, $\mu_{min}(a_i)$, across all M compute node completion time distributions. Each of these distributions is obtained by mapping a_i to the compute node and determining the new completion time distribution for the compute node. The worth of application a_i, denoted as $\eta(a_i)$, is then determined as a result of the following normalization

$$\eta(a_i) = \frac{\mu_{min}(a_i)}{\sum_{\text{unmapped } a_k} \mu_{min}(a_k)}. \quad (5)$$

In Phase 2, an unmapped application is stochastically selected (procedure described later) and assigned to its $m_{best}(a_i)$ compute node. The ant procedure is repeated until all applications are mapped.

Let the fitness of ant s, denoted as $f(s) \in (0, 1)$, be determined as a rank of ant s in the sorted order of ants in the current iteration. Sorting is based on the minimum possible level of Λ, obtained with a PMR call invoked at the end of each ant procedure, while ranking employs a concept of linear bias function [30]. The pheromone table is updated at the end of each high-level iteration, i.e., when all ants complete their paths. Specifically, if ρ denotes a coefficient that represents pheromone evaporation, B_s denotes the set of application-compute node assignments comprising the path of ant s, and assuming Q ants released, each $\tau(a_i, \eta_j)$ is updated as follows

$$\tau(a_i, \eta_j) = \rho \times \tau(a_i, \eta_j) + \sum_{s=1}^{Q} f(s) \times \mathbf{1}(a_i \text{ assigned to } \eta_j \text{ in } B_s). \quad (6)$$

Initially, all values in the pheromone table were set to 1.

Let α be the scalar that controls the balance between the pheromone value and worth. The probability that ant s selects application a_i to be mapped next is

$$\mathbb{P}[a_i \text{ selected next}] = \frac{\alpha \times \tau(a_i, m_{best}(a_i)) + (1-\alpha) \times \eta(a_i)}{\sum_{\text{unmapped } a_k} \alpha \times \tau(a_k, m_{best}(a_k)) + (1-\alpha) \times \eta(a_k)}. \quad (7)$$

The scalar α was determined experimentally by incrementing from 0 to 1 in 0.1 steps. In the simulation trials tested, the performance peak was detected with α equal to 0.5. The pheromone evaporation factor ρ of 0.01 was determined in a similar manner. The total number of ants for each iteration was set to 50; any further increase of this number in the experiments resulted in performance degradation. Note that numerical values for all the aforementioned parameters were determined with respect to the input specified for the conducted experiments—i.e., these values must be readjusted for different inputs.

```
initialize pheromone table;
while CSC not met
    for each ant
        while there are unmapped applications
            select application a_i according to (7);
            map application a_i to m_best(a_i) compute node;
            break ties arbitrarily;
        end of while;
        compute f(s) via PMR call;
    update pheromone table according to (6);
end of while
map each application a_i to its m_best(a_i) compute node.
```

Figure 4. The Ant Colony Optimization procedure.

4.4 Simulated Annealing

The Simulated Annealing (SA) algorithm (summarized in Fig. 5) —also known in the literature as Monte Carlo an-

nealing or probabilistic hill-climbing [20]— is based on an analogy taken from thermodynamics. In SA, a randomly generated solution, structured as the chromosome for GA and used as a startup point, is then iteratively modified and refined. Thus, SA in general, can be considered as an iterative technique that operates with one possible solution (i.e., resource allocation) at a time.

To deviate from the current solution in an attempt to find a better one, SA repetitively performs the mutation operation in the same fashion as for GA including local search. Once a new *unique* solution, denoted as S_{new}, is produced (SA uses the same procedure as GA to determine uniqueness), a decision regarding the replacement of a previous solution with a new one has to be made. If the quality of a new solution, $\Lambda(S_{new})$, found after evaluation, is higher than the old solution, the new solution replaces the old one. Otherwise, SA uses a procedure that probabilistically allows poorer solutions to be accepted during the search process, which makes this algorithm different from other strict hill-climbing algorithms [20]. This probability is based on a system temperature, denoted as $\mathbb{T}$, that decreases with each iteration. As the system temperature "cools down" it becomes more difficult for poorer solutions to be accepted. Specifically, in the latter case, the SA algorithm selects a sample from the range $[0, 1)$ according to the uniform distribution. If

$$random[0,1) > \frac{1}{1+\exp^{(\frac{\Lambda(S_{old})-\Lambda(S_{new})}{\mathbb{T}})}} \qquad (8)$$

the new poorer resource allocation is accepted; otherwise, the old one is kept. As it follows from (8), the probability for a new solution of similar quality to be accepted is close to 50%. In contrast, the probability of poor solutions to be rejected is rather high, especially when the system temperature becomes relatively small.

After each mutation (described in the GA procedure) that successfully produces a new unique solution, the system temperature $\mathbb{T}$ is reduced to 99% of its current value. This percentage, defined as a cooling rate, was determined experimentally by varying the rate in the range of $(0.9, 1]$ in 0.01 steps. The initial system temperature in (8) was set to the Λ of the chosen startup resource allocation.

```
S_old ← initial randomly generated resource allocation;
𝕋 ← Λ(S_old);
while CSC not met
    S_new ←result of successful mutation;
    if Λ(S_new) < Λ(S_old)
        S_old ← S_new;
    else if (8) holds
            S_old ← S_new;
𝕋 ← 0.9 × 𝕋;
end of while
```

Figure 5. The Simulated Annealing procedure.

5 Lower Bound Calculation

To evaluate the absolute performance attainable by the developed resource allocation techniques, a lower bound (LB) on the minimum period Λ was derived based on the assumption that the specified level of the stochastic robustness metric is greater than or equal to 0.5, i.e., $\theta \geq 0.5$, which is typical for practical implementations. The process of calculating LB involves two major steps. In the first step, a "local" lower bound on Λ is established for a given mapping. In the second step, a unique LB is computed for all possible local lower bounds by solving a relaxed form of the Integer Linear Program formulated for the resource allocation problem.

Step 1: Consider a *given* complete resource allocation of N applications on M compute nodes. Let $\overline{\Lambda}$ denote the maximum of the means across all M completion time distributions, $\underline{\mu(\sum_{i=1}^{n_j} T_{ij})}$, i.e., $\overline{\Lambda} = \max\{\mu(\sum_{i=1}^{n_j} T_{ij}) \mid 1 \leq j \leq M\}$. As an assumed level of the stochastic robustness metric is greater than or equal to 0.5, $\overline{\Lambda}$ represents the smallest possible time period for a given mapping. To observe this, recall that

1. mean $\mu(a)$ is a "center of mass" of the distribution of random variable $\underline{a}$, so that if $\underline{z}$ is the compute node given by $z = \text{argmax}\{\mu(\sum_{i=1}^{n_j} T_{ij}) \mid 1 \leq j \leq M\}$, then $\mathbb{P}[\psi_z \leq \overline{\Lambda}] = 0.5$;

2. $\mathbb{P}[\psi_z \leq \overline{\Lambda}] \geq \mathbb{P}[\psi \leq \overline{\Lambda}]$ because according to (2), $\mathbb{P}[\psi \leq \overline{\Lambda}]$ is computed as an M-product of $\mathbb{P}[\psi_j \leq \overline{\Lambda}]$, where each of M terms is less than or equal to one.

Step 2: An objective here is to determine LB, denoted as $\underline{\Lambda^*}$, such that $\Lambda^* \leq \min\{\overline{\Lambda} \mid \text{ all possible mappings}\}$. Relying on the property that the sum of means is equal to the mean of the sums, i.e., $\sum_{i=1}^{n_j} \mu(T_{ij}) = \mu(\sum_{i=1}^{n_j} T_{ij})$, the problem of finding Λ^* can be formulated in the following Integer Linear Programming (ILP) form[2].

Let a *binary* decision variable $\underline{x[i,j]} \mid \{1 \leq i \leq N; 1 \leq j \leq M\}$ be equal to one if application a_i is assigned to compute node η_j, and equal to zero if a_i is not assigned to compute node η_j. The ILP objective function can be stated as

$$\text{minimize} \quad \Lambda^* = \max\{\sum_{i=1}^{N} \mu(T_{ij}) \times x[i,j] \mid 1 \leq j \leq M\}.$$

The objective function is subject to conditions (a) and (b):

$$x[i,j] \in \{0,1\} \quad \text{for } 1 \leq i \leq N, \quad 1 \leq j \leq M; \qquad \text{(a)}$$

$$\sum_{i=1}^{N} x[i,j] = 1 \quad \text{for } 1 \leq j \leq M; \qquad \text{(b)}$$

In addition to condition (a) explained above, condition (b) forces each application to be mapped to the system. For small-scale problems, a global optimal solution can be found for the derived ILP form in a reasonable time (e.g., by applying Branch-and-Bound technique). However, condition (b) makes the ILP form NP-complete [21], so that for large-scale problems a Linear Programming (LP) relaxation is required to the ILP form that implies that condition (a) is relaxed to real numbers, i.e., $x[i,j] \in [0,1] \mid \{1 \leq i \leq N, 1 \leq j \leq M\}$. Due to this relaxation, in general, an LP solution does not correspond to a valid mapping, but

[2]The presented below ILP formulation can easily be converted to a canonical ILP form.

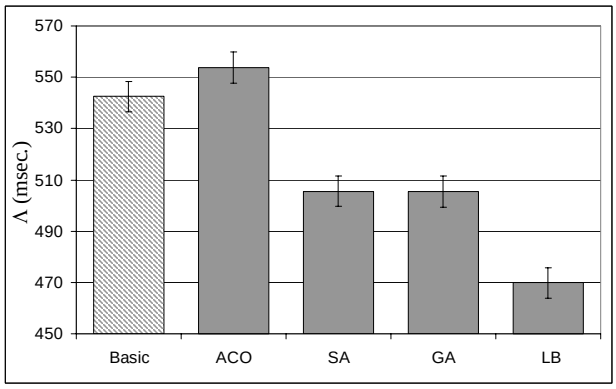

Figure 6. A comparison of the results obtained for the described heuristics. The y-axis corresponds to a minimum Λ value achieved by each heuristic. The error bars demonstrate 95% confidence intervals computed over 50 trials.

it allows the global optimal solution to be found in *polynomial* time [11], that will be a lower bound for the ILP global optimal solution Λ^*. Also note that the derived LB is getting tighter for the stochastic robustness levels approaching 0.5; this is a result of using mean values in the LB computation.

6 Experimental Results

The results of the simulation are presented in Fig. 6. Both the GA and SA heuristics were able to improve upon the results of the Basic heuristic of [25] by more than 7% with respect to the absolute performance and by 50% with respect to the derived LB. However, the ACO procedure was unable to improve upon the results of the Basic heuristic but was able to produce a results such that the confidence intervals of the ACO and Basic results were overlapping.

For the 50 trials tested, LB produced a mean minimum supportable Λ of 469.8 msec. The mean of the Basic heuristic over the same 50 trials was found to be 542.5 with a 95% confidence interval of 7.07. The ACO results had a mean minimum supportable Λ value of 553.7 with a 95% confidence interval of 6.2. The SA procedure for the same trials produced a mean Λ value of 505.6 with a 95% confidence interval of 5.9. The GA result was very similar to the SA result, producing a mean Λ value of 505.3 with a 95% confidence interval of 6.1.

Both the GA and SA heuristics performed comparably in this simulation environment. The success of the SA procedure and the near overlap of the SA and GA results may suggest that the local search procedure used in the mutation operator by both GA and SA is responsible for their marked improvement over Basic. Additional experiments were conducted with the GA without utilizing local search and although the simple GA was able to improve the average result of the Basic heuristic by almost 2% the improvement was not statistically significant. The GA with the local search mutation operator outperforms the GA without local search by a significant margin.

The ACO heuristic was unable to improve upon the results of the Basic heuristic of our previous work. This might suggest that using only the mean values of the execution time distributions to construct solutions in Phase 1 is insufficient. Instead of operating with mean values, intermediate minimum levels of Λ could be computed through PMR calls to potentially improve the results of the ACO procedure. However, this would dramatically increase the number of evaluations required by ACO to produce the ants of each iteration. In so doing, the number of high-level iterations that the ACO procedure would be able to complete within the CSC would be significantly reduced. The major hindrance to the effectiveness of ACO in this environment is that it relies on the repetitive application of a constructive heuristic within an iteration to update the pheromone table. As shown in [25], constructive heuristics such as the Basic heuristic require a large number of time-consuming FFT executions, this approach significantly slows down each ant's production of a completed resource allocation, which, in turn, limits the number of high-level iterations that can be performed within the CSC.

The success of combining a simple local search with GA and SA suggest that a more exhaustive local search may be worth investigating in future work. The more exhaustive local search might consider swapping applications between compute nodes in addition to moving applications between compute nodes. Although the introduction of swapping will increase the number of evaluations required to complete the local search procedure, it may lead to an improved result over the current coarse approach to local search.

7 Related Work

In HC systems the concept of robust resource allocation 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 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 [24]. 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, 18]. 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 [14], 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 heuristic approaches presented in [27]. 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. A variety of iterative algorithms demonstrate higher performance as compared to the non-iterative greedy heuristics. However in the deterministic domain, 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.

Genetic algorithm was adopted to address resource allocation problem in distributed systems in [28], assuming task execution times deterministic and known a priori. Later, 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. 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 iterative resource allocation algorithms.

8 Conclusion

This paper presented a set of iterative algorithms for finding stochastically robust resource allocations in heterogeneous clusters where workload surges are likely to occur. The objective function in algorithm design was based on our stochastic robustness metric, which 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. Given the raw volume of computation required to compute the proposed robustness metric in the stochastic domain, the FFT method and bootstrap approximation technique were outlined along with the underlying assumptions of independence.

Multiple parameters pertaining to each iterative algorithm were setup for the best performance in the simulated environment that replicated a heterogeneous cluster-based processing center of a radar system. The goal in the experiments was to generate a resource allocation that allows for the minimum time period between sequential sensor outputs and guarantees a specified probability level that data processing is completed in time.

The performance analysis of multiple test trials revealed a great potential of the GA and SA algorithms to efficiently manage resource allocation in a large class of heterogeneous clusters operating on periodic data sets. In addition to complex radar systems, the developed resource allocation techniques can be applied in other practical implementations such as surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems.

Acknowledgement

The authors would like to thank Jennifer Hale, Patrick Moranville, and Robert Umland for their contributions.

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] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

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

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

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: MIT Press, 1992.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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

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

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

[23] 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.

[24] 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 *Proceedings International Conference on Parallel Processing (ICPP–06)*, Aug. 2006, pp. 459–467.

[25] V. Shestak, J. Smith, R. Umland, J. Hale, P. Moranville, A. A. Maciejewski, and H. J. Siegel, "Greedy approaches to static stochastic robust resource allocation for periodic sensor driven distributed systems," in *Proceedings the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06)*, June 2006, pp. 3–9.

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

[27] 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.

[28] 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.

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

[30] 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.

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

[32] T. Yamada and C. Reeves, "Permutation flowshop scheduling by genetic local search," in *Proceedings Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sept. 1997, pp. 232–238.