# Models and Heuristics for Robust Resource Allocation in Parallel and Distributed Computing Systems

David L. Janovy[1], Jay Smith[1,3], Howard Jay Siegel[1,2], and Anthony A. Maciejewski[1]

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

[3]IBM
6300 Diagonal Highway
Boulder, CO 80301
bigfun@us.ibm.com

## Abstract

*This is an overview of the robust resource allocation research efforts that have been and continue to be conducted by the CSU Robustness in Computer Systems Group.*

*Parallel and distributed computing systems, consisting of a (usually heterogeneous) set of machines and networks, frequently operate in environments where delivered performance degrades due to unpredictable circumstances. Such unpredictability can be the result of sudden machine failures, increases in system load, or errors caused by inaccurate initial estimation. The research into developing models and heuristics for parallel and distributed computing systems that create robust resource allocations is presented.*

## 1. Introduction

Parallel and distributed computing systems must often operate in environments replete with uncertainty while continue to provide a required level of service. Designing robust systems for such environments involves determining resource allocations that can account for uncertainty in estimated system parameters.

Our research into robust resource allocations has progressed in several stages. This paper provides an overview of this research into robust resource allocations in parallel and distributed computing systems. Section 2 discusses the deterministic models of robustness and related heuristics that have been investigated. The stochastic robustness metric developed as part of this ongoing research is presented in Section 3 along with a selection of the resource allocation heuristics developed for static stochastic resource allocation environments. Our most recent stochastic research explores the design of robustness metrics in stochastic dynamic environments and is discussed in Section 4.

This research was supported by NSF under grant No. CNS-0615170 and by the Colorado State University George T. Abell Endowment.



## 2. Deterministic Robustness

The following is a brief description of the deterministic robustness research in heterogeneous parallel and distributed computing systems that has been conducted by the CSU Robustness in Computer Systems Group.

A mathematical description of a general robustness metric when the system parameters are represented by deterministic estimates is introduced in [1]. Also in [1], the *FePIA* procedure is developed for deriving an instantiation of the robustness metric for arbitrary given resource allocation environments. Examples of how the *FePIA* procedure can be applied in different situations are shown, as well as the utility of the performance metric is introduced.

Creating robust resource allocations when there is uncertainty in task execution times is studied in [10]. Two variations of this problem are considered. The first variation considers robust static allocation of resources with a constraint on the total time allowed to complete all of the tasks on a fixed set of machines. The goal is to maximize the collective allowable error in execution time estimates. The second variation analyzes the purchasing choices of a set of machines that are to comprise a heterogeneous system. Dollar cost constraints are imposed and the goal is to select a set of machines that maximizes the robustness of the resource allocation while meeting the completion time constraint.

In [4], data sets arriving from a space-based weather monitoring system must be processed before the next data

set arrives. Specific tasks that operate on the data sets have high, medium, or low priorities. The arrival time of the next data set is uncertain and the goal is to have all the high priority tasks and as many of the medium and low priority tasks completed before the next data set arrives. Medium and low priority tasks are assigned a weight that is combined with a value representing the "likelihood" that that task will complete and is used to determine the "worth" of a resource allocation. Static resource allocation heuristics for this environment are designed and compared. The heuristics execute in two phases: 1) minimize the makespan of the high priority tasks and 2) maximize the overall worth of the resource allocation for medium and low priority tasks.

In [2], greedy static resource allocation heuristics are developed for a heterogeneous set of sensors, applications, machines, networks, and actuators. In this environment, the application tasks are continuously executing, receiving new input data sets from the sensors and producing corresponding new control commands for the actuators. The heuristics try to create a robust initial allocation, i.e., one that can withstand the maximum increase in workload, generated by changes in the sensor data sets, until a runtime reallocation of resources is required to continue meeting a given set of constraints. If a heuristic cannot determine a resource allocation that meets the constraints, it is said to have failed. Developing heuristics that both allow increased workload and have low failure rates is the focus of [2].

Robust static resource allocations in distributed, multitasking computing systems aboard shipboard environments is studied in [5]. Complete and partial resource allocation schemes were considered. The complete resource allocation scheme is used when the system has enough resources to accommodate all tasks and the partial method is used when the resources are limited or service level constraints are violated. Each task is assigned a "worth" factor that is used to determine the total worth of a resource allocation. The goal of the developed heuristics is to create a robust initial resource allocation that maximizes the total worth of the system's performance and the system's capacity to absorb unpredictable increases in input workload without quality of service violations.

A model is developed in [3] for quantifying robustness in a dynamic heterogeneous environment, where task arrival times are not known *a priori* and actual task execution times may deviate from the estimated execution times. Two problem variations are considered. The first is a robustness constrained analysis, where the goal is to minimize the makespan while maintaining a certain level of robustness. The second is a makespan constrained analysis, where the robustness is maximized while the makespan remains below a specified limit.

The next section discusses stochastic robust resource allocation (as opposed to deterministic model based) for heterogeneous parallel and distributed computing systems. This stochastic model robustness research builds upon the work of the previously discussed deterministic robustness research.

## 3. Stochastic Robustness

### 3.1. Model

This section summarizes the definition of the stochastic robustness metric (SRM), that is part of the research described in [7]. Heterogeneous parallel and distributed computing systems are subject to uncertainties that may lead to variations in application execution times. For a computing system that consists of $M$ nodes, let $n_j$ be the number of applications assigned to compute node $c_j$ and let random variable $T_{ij}$ denote the execution time of each individual application $a_{ij}$ on compute node $c_j$. 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. A characterization of system performance referred to as the performance characteristic $\psi$, 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 $\psi$ is itself a random variable.

The time period $\Lambda$ between arriving data sets is held fixed, limiting the acceptable range of possible variation in system performance, i.e., $\psi \leq \Lambda$. **The stochastic robustness metric, denoted by $\theta$, is the probability that the makespan of the system does not exceed $\Lambda$, 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 arriving data sets. Clearly, unity is the most desirable stochastic robustness metric value, i.e., there is zero probability that the system will violate the established time period constraint.

In the model of compute node $c_j$, the functional dependence between the set of local uncertainty parameters $\{T_{ij} \mid 1 \leq i \leq n_j\}$ and the local performance characteristic $\psi_j$ 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 of a resource allocation can be found as the product of the probabilities that execution on each compute node satisfies the imposed $\Lambda$ time period. Mathematically, this is given as

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

If the execution times $T_{ij}$ for applications mapped to compute node $c_j$ are mutually independent, then each multiplication term in Equation 2 can be computed using an $(n_j - 1)$-fold convolution of probability mass functions (pmfs) $f_{T_{ij}}$

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

In the next subsection, we present heuristics that utilize the stochastic robustness metric during resource allocation. These heuristics apply the SRM as a constraint on the resource allocation instead of focusing on minimizing the time period $\Lambda$. That is, the heuristics try to minimize $\Lambda$ while ensuring that the stochastic robustness remains greater than or equal to a given value.

### 3.2. Heuristics

In [6] and [8] we study heuristics that attempt to minimize $\Lambda$ for a given environment. A two-phase greedy heuristic (BASIC) from our research in [8] and an iterative heuristic (Steady State Genetic Algorithm) from [6] are summarized in the following subsections. Both heuristics use a time period minimization routine (PMR) that is described next.

**Period Minimization Routine.** The PMR procedure determines the minimum possible value of $\Lambda$ 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 for each compute node corresponding to the completion time (i.e., $\sum_{i=1}^{n_j} T_{ij}$) distribution expressed in a pmf form. The completion time pmf for compute node $c_j$ 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 $\Lambda$ 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. The PMR procedure is summarized in Figure 1.

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

**Figure 1. The Period Minimization Routine procedure.**

After $\Omega$ 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 $\theta$ is strictly increasing as the number of impulses considered for its computation grows.

**Greedy Heuristic.** The two-phase greedy heuristic (BASIC) [8] is based on the principles of the Min-Min algorithm. The heuristic traverses through $I$ 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 notation $\Lambda(a_i, c_j)$ will be used to denote a PMR call that returns the minimum value of $\Lambda$ for the specified level of stochastic robustness when application $a_i$ is added to compute node $c_j$. The BASIC procedure is summarized in Figure 2.

**Iterative Heuristic.** The adapted genetic algorithm (GA) implementation is summarized in Figure 3. Each chromosome in the GA models a complete resource allocation as a vector of numbers 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 initial members of the population were generated by applying the one-phase sorting greedy heuristic presented in [8], in which the application ordering was perturbed to produce different resource allocations to serve

```
while not all applications are mapped
  for each unmapped application a_i
    find the compute node c_j such that
    c_j ← argmin{Λ(a_i, c_j) | 1 ≤ j ≤ M};
  from all (a_i, c_j) pairs found above
  select the pair(s) (a_x, c_y) such that
  (a_x, c_y) ← argmin{Λ(a_i, c_j) | all (a_i, c_j)};
  map a_x on c_y;
end of while
```

**Figure 2. Summary of the BASIC two-phase greedy procedure.**

```
generate initial population;
evaluate each chromosome;
rank population based on Λ values calculated using PMR;
while stoping criteria not met
  generate new chromosomes using crossover
    and mutation;
  insert unique offspring into population based on Λ
    value calculated using PMR;
  drop worse chromosomes to maintain population size;
end of while
output the best solution.
```

**Figure 3. The steady state Genetic Algorithm procedure.**

as the initial members of the population. In addition, the solution produced by the BASIC heuristic from [8] was also added to the initial population.

The GA 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 a linear bias function where the population of chromosomes is sorted according to evaluation of $\Lambda$ values. The most fit chromosome corresponds to a resource allocation with the smallest $\Lambda$ value supportable at the specified level of stochastic robustness $\theta$. 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.

## 4. Robustness of Resource Allocation Heuristics in a Stochastic Dynamic Environment

We present the use of a stochastic robustness metric [7] to quantify the robustness of a resource allocation in a *dynamic* environment. This research focuses on an instance of a dynamic, heterogeneous computing (HC) system where task arrival times are not known in advance and exact task execution times are uncertain prior to their completion. All incoming tasks to the system are assumed to have been previously classified into gross classifications based on their relative complexity. Each task class defines a set of pmfs, where each pmf describes the probability of all execution times for that class on a given machine within the HC suite. Further, all of the classes of tasks that the system may be asked to perform are known in advance. Each arriving task has a deadline, relative to its arrival time established in advance that limits its total processing time. Although some tasks may belong to the same class, task execution times may still vary depending on the details of the submitted task. For this reason, task execution times are modeled as random variables and the probability distributions describing task execution times are assumed known. The impact of missing a task deadline is modeled by a constant factor penalty for each task deadline missed. That is, the overall performance objective of a resource allocation heuristic in this environment is to minimize the number of tasks that miss their deadlines.

Following from our prior work [7], the robustness of the finishing time for a task can be found as the probability that the task will finish before its deadline. This probability defines a local robustness characteristic. Individual local robustness characteristics for all tasks currently awaiting execution, can be combined to produce an instantaneous SRM, at a given mapping event by taking the product of the local robustness characteristics. Mapping events occur within the system whenever a new task arrives or an existing task completes.

An instantaneous SRM value is generated at each mapping event during the course of a dynamic resource allocation and is used to define a dynamic SRM value. The dynamic SRM value is defined as the average of the instantaneous SRM values found during a given time period. Given the relationship between the dynamic SRM value and the performance metric, the dynamic SRM value can be used to predict the relative performance of two resource allocation heuristics. That is, if a heuristic consistently maintains a high instantaneous robustness value over some number of mapping events, then there is a consistently low probability that tasks will miss their deadlines over that same period.

In a dynamic environment, the set of tasks being considered is constantly changing due to task arrivals and completions. Computing the instantaneous SRM value at a given mapping event requires that the start time of the currently executing task on each machine is known. Determining the start time for a task $i$ requires knowledge of the actual execution time of the previously executed task $k$ on that machine to calculate task $k$'s actual completion time. During

the initial set of simulations used for heuristic evaluation, our methodology utilizes the expectation of the execution time pmf as the actual execution time for each task. Thus, the start time of the subsequent task $i$ is known, enabling the calculation of instantaneous SRM value.

Because of variations in the set of tasks to be executed and changes in their arrival ordering, multiple simulation trials should be conducted to adequately predict the typical relative performance among the resource allocation heuristics. In evaluating our results [9], we demonstrate that in a dynamic environment a small number of simulation trials are required to sufficiently indicate the performance of a resource allocation heuristic relative to our given performance objective.

The dynamic SRM values for all simulation trials are then combined by taking their average to determine a single dynamic SRM value for the resource allocation heuristic. The dynamic SRM values for different resource allocation heuristics can then be compared to select the approach that is more robust within the given environment [9].

## 5. Conclusions

In our research on robust resource allocation we try to investigate three questions for any given environment. The first is, what behavior makes the system robust? The second is, what uncertainty must the system be robust against? The third is, how can one quantify exactly how robust the resource allocation is?

We have examined these three robustness questions using two different paradigms: deterministic and stochastic. In Section 2, we gave pointers to the research where we define the deterministic robustness measure and describe environments for which we have developed both static and dynamic resource allocation heuristics based on this measure. Our stochastic model of robustness was defined in Subsection 3.1 and static resource allocation heuristics based on this model were presented in Subsection 3.2. Finally, in Section 4 we summarized how the stochastic model can be used to quantify robustness when doing dynamic resource allocation.

## References to Research Conducted by the CSU Robustness in Computer Systems Group


[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*, 15(7):630–641, 2004.

[2] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Robust resource allocation for sensor-actuator distributed computing systems. In *2004 International Conference on Parallel Processing (ICPP 2004)*, pp. 178–185, Aug. 2004.

[3] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye. Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. *Journal of Supercomputing*. accepted, to appear.

[4] M. Oltikay, J. Brateman, J. White, J. Martin, K. Knapp, A. A. Maciejewski, and H. J. Siegel. Robust resouce allocation in weather data processing systems. In *$8^{th}$ Workshop on High Performance Scientific and Engineering Computing (HPSEC 2006)*, in the *Proceedings of the 2006 International Conference on Parallel Processing Workshops*, pp. 445–454, Aug. 2006.

[5] V. Shestak, E. K. P. Chong, A. A. Maciejewski, H. J. Siegel, L. Bemohamed, I.-J. Wang, and R. Daley. Resource allocation for periodic applications in a shipboard environment. In *$14^{th}$ Heterogeneous Computing Workshop (HCW 2005)*, Apr. 2005.

[6] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski. Iterative algorithms for stochastically robust static resource allocation in periodic sensor driven clusters. In *$18^{th}$ IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*, pp. 166–174, Nov. 2006.

[7] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski. A stochastic approach to measuring the robustness of resource allocations in distributed systems. In *2006 International Conference on Parallel Processing (ICPP 2006)*, pp. 459–467, Aug. 2006.

[8] 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 *2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2006)*, volume 1, pp. 3–9, June 2006.

[9] J. Smith, L. D. Briceno, A. A. Maciejewski, H. J. Siegel, T. Renner, V. Shestak, J. Ladd, A. Sutton, D. Janovy, S. Govindasamy, A. Alqudah, R. Dewri, and P. Prakash. Measuring the robustness of resource allocations in a stochastic dynamic environment. In *$21^{st}$ International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007. accepted, to appear.

[10] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horuichi, 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*. accepted, to appear.