Heuristics for Robust Allocation of Resources to Parallel Applications with Uncertain Execution Times in Heterogeneous Systems with Uncertain Availability

Timothy Hansen, Florina M. Ciorba, Anthony A. Maciejewski,
Howard Jay Siegel, Srishti Srivastava, and Ioana Banicescu

***Abstract*—The scheduling of moldable parallel applications to clusters of processors is challenging, where the number of processors on which a moldable application executes is decided by the scheduler. When the application execution times are stochastic in nature, and the availability of the resources is uncertain, this becomes an even greater challenge. A model is presented for the stochastic execution times of moldable parallel applications that are assigned to heterogeneous parallel resources, incorporating the change in execution times when applications are mapped to different numbers of processors. To account for the uncertainties in both application execution times and resource availability, a robustness model that combines the two sources of uncertainties is proposed. Using this robustness model, three novel iterative-greedy heuristics are developed to allocate heterogeneous resources to batches of parallel applications to maximize the probability of completing by a designated time, called the makespan goal. To verify the performance of the proposed heuristics, a simulation study is conducted using different batch and system sizes. To showcase the benefit of using the proposed iterative-greedy heuristics, their performance is studied against two comparison heuristics. The five heuristics are evaluated against the upper bound on robustness.**

***Index Terms*—heterogeneous systems, heuristic optimization, moldable parallel applications, robustness, stochastic resource allocation.**

I. Introduction

TODAY'S computing systems are often heterogeneous in nature, comprised of machines with differing computational capabilities to satisfy the diverse computational requirements of applications [1], [2]. For example, a mixture of general purpose and programmable digital machines, along with application-specific systems-on-a-chip, were shown to solve parallel jobs with real-time constraints [3], [4]. The scheduling of applications on such heterogeneous systems has been shown, in general, to be NP-complete [5]. Scheduling decisions become more difficult in systems with uncertain processor availability (this can be due to system jitter/noise [6], or the time sharing of resources [7]) and with application execution times that are modeled as stochastic due to uncertain input data [8].

Manuscript received January, 2014; revised February, 2014. This work is supported by the National Science Foundation (NSF) under grant numbers CNS-0905399, CCF-1302693, and IIP-1034897; the Colorado State University George T. Abell Endowment; and the German Research Foundation in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing." This research utilized the CSU ISTeC Cray HPC System supported by NSF Grant CNS-0923386.

T. Hansen, A. A. Maciejewski, and H. J. Siegel are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA. e-mail: {timothy.hansen,aam,hj}@colostate.edu.

F. M. Ciorba is with the Center for Information Services and High Performance Computing at Technische Universität Dresden, 01062 Dresden, Germany. e-mail: florina.ciorba@tu-dresden.de.

S. Srivastava and I. Banicescu are with the Department of Computer Science and Engineering, Mississippi State University, Mississippi State, MS 30692, USA. e-mail: {ss878@,ioana@cse.}msstate.edu.

We consider a batch of scientific *moldable* parallel applications with stochastic execution times, where a moldable parallel application is one that differs in execution time as a function of the numbers of processors (determined by the scheduler) on which it executes. These applications need to be allocated resources from a set of heterogeneous processor types, where the processor types are heterogeneous in both their computational capabilities (affecting the application execution times) and the number of processors available. All applications in the batch should be scheduled in such a way as to finish executing close to a given *makespan goal*.

To allocate resources to applications, we propose a new batch scheduler. The batch scheduler must allocate resources in the presence of the two uncertainties of application execution times and system availability. To minimize the impact of the two sources of uncertainty on achieving the makespan goal, our resource allocations should be robust against these uncertainties. To accomplish this goal, we introduce a model that combines the impact on performance of two sources of uncertainties into a single performance metric of *robustness* [9], where we define robustness as the probability that a batch of applications finishes by the given makespan goal. Three iterative-greedy resource allocation heuristics that use this measure of robustness were designed. The allocation decisions that need to be made for each application are: (1) on what processor type to run, and (2) on how many processors of a given type to run. We design our resource allocation heuristics to maximize robustness by using stochastic knowledge of the uncertain execution times and uncertain system availability to intelligently allocate processors to applications.

This paper is based on the first stage of the dual-stage optimization framework introduced in [10]. In the first stage, which is the focus of this paper, a batch of applications is allocated resources from a set of heterogeneous processor types. The second stage, which is not part of this paper, performs fine-grain runtime optimization for each application given the allocated resources from the first stage. The system and the flow of information is shown in Fig. 1. In this paper, we design and evaluate novel resource allocation heuristics (no heuristics presented in [10]). The heuristics presented here utilize a more realistic parallel execution time model and a much larger system than those discussed in [10].

Related prior work on resource allocation and scheduling for heterogeneous systems has occurred in the areas of heuristic optimization and modeling. Uncertainties in application execution times were taken into account in [11]–[13]. These uncertainties lead to robustness as a performance measure (e.g., [8], [9]). The uncertainty in the availability of computing resources was studied as system noise [6] and operating system (OS) overhead [14]. Iterative-greedy

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

heuristics have been shown to perform well for scheduling problems [15] and balanced resource allocation techniques are often used in practice [16]. Our work differs from previous work in that we design new heuristics that take into account the uncertainty in system availability as well as the uncertainty in application execution times.

In this paper, we make the following contributions:

- The design of a model for moldable parallel applications with stochastic execution times running in a heterogeneous computing environment.
- A new robustness model and measure dealing with, and combining, two sources of stochastic uncertainties for use in the resource allocation of processors to applications and the performance evaluation of said resource allocations.
- The design and analysis of three novel iterative-greedy heuristics across three different platforms of a varying number of processor types compared to two reference heuristics and an upper-bound.

The rest of the paper is organized as follows. The system model is described in Section II. In Section III, the developed heuristics are presented in detail. The parameters used for the analysis are given in Section IV with the analysis results being shown in Section V. Finally, concluding remarks and a brief description of future work are summarized in Section VI.

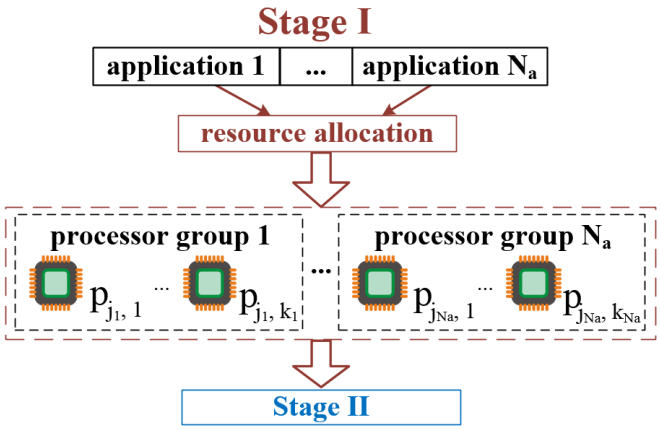

Fig. 1. Dual-stage optimization framework with a focus on Stage I. In the first stage, a batch of N_a scientific moldable parallel applications are allocated resources from heterogeneous processor types according to a given resource allocation heuristic. In Stage II, a runtime optimization is performed for each application using the allocated resources from Stage I.

II. System Model

A. Batch Scheduler

The proposed model for the batch scheduler is given in Fig. 2. There are three times of interest shown: the time the assignment of resources for batch z starts (t_0), the time batch z starts executing (t_1), and the time that batch z finishes executing (t_2). Thus, the resource allocation heuristic executes from t_0 to t_1. Without a loss of generality, we are assuming a workload of one batch so the subscript denoting the batch number will be dropped from the notation. This research is applicable for any number of subsequent batches.

Given a batch of N_a moldable parallel applications, a scheduling heuristic is used to allocate computing resources to each application. The applications are assumed to be independent and without precedence constraints. The allocation decision that needs to be made for application i is

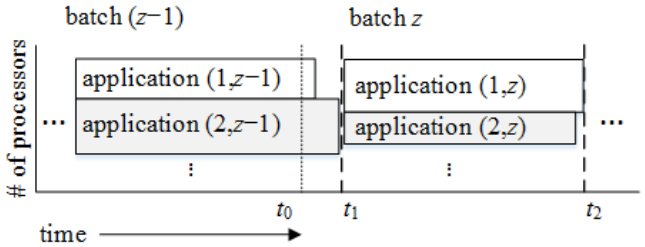

Fig. 2. Proposed batch scheduler model. At some time t_0, the applications in batch z will be assigned resources using a given heuristic. At t_1, the last application of batch $(z-1)$ finishes executing and batch z can begin executing using the resources allocated by a given heuristic. Time t_2 denotes when batch z finishes executing and batch $(z+1)$ begins executing, *ad infinitum*.

twofold. First, application i must be assigned to one of N_p processor types (denoted processor type j). Second, the application must be allocated a number of processors, k, of processor type j. Let $\boldsymbol{I}$ be a vector of length N_a representing a complete resource allocation, where the i^{th} entry is a (j,k) tuple that represents application i being assigned to k processors of type j (i.e., $I[i] = (j_i, k_i)$).

To avoid fragmentation of the system resources, all N_a applications start executing at the same time (i.e., each batch of applications can leverage the ability of the entire set of system resources), shown as t_1 in Fig. 2. In addition to avoiding fragmentation, other scenarios in which this holds true are in scatter-gather operations and MapReduce [17]. Because all applications in the batch start executing at the same time, the next batch of applications cannot start executing until the current one is finished (t_1 and t_2 in Fig. 2). This implies that the scheduling heuristics should try to allocate resources so applications finish at approximately the same time to prevent the waste of system resources (i.e., idle machines).

B. Applications

1) Heterogeneity: The execution time for each application i in the batch of applications differs across heterogeneous processor types. For a fixed number of processors, if processor type A is faster than processor type B for a given application, it is not necessarily true that processor type A is faster than processor type B for *all* applications. We assume the application execution time distributions are known *a priori*. This information, in practice, can be obtained by analytical, historical, or experimental techniques [8], [18].

2) Parallel Model: We use Downey's parallel speedup model [19] to describe how the execution times of real applications change as a function of the number of processors allocated. Given k processors, the speedup of an application is denoted $S(k)$. If $S(k) = x$ for a given resource allocation, application i will execute x times as fast in parallel on k processors ($k > 1$) of type j than if it was run serially (i.e., $k = 1$). This model takes into account the different finishing times of each processor and the execution time of the application is determined by the longest running processor.

C. Uncertainties

1) Application Execution Time: Each application on a given processor type has an uncertain execution time, for example because of differing input data. Because the system is

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

modeled as heterogeneous, each application has a probability distribution describing its execution time on each processor type. These execution time distributions are assumed to describe the *serial execution times* of the applications. Let $\boldsymbol{\mathcal{T}_{i,j}}$ be a random variable describing the serial execution time of application i on processor type j. To obtain the *parallel execution time distribution*, we use the Downey model by scaling the time axis of the probability distribution of $\mathcal{T}_{i,j}$ by $\frac{1}{S(k)}$.

2) System Slowdown: In addition to the uncertainty in application execution times, the system availability of each processor type is uncertain. The set of system resources available are N_p *processor types*, where processor type j has $\boldsymbol{n_j}$ processors. Each processor type is assumed to have an associated *system slowdown* (defined as the reciprocal of the system availability). This could be due to OS jitter or system noise [6], or the time sharing of resources [7].

Because the slowdown of a processor type is uncertain, we model slowdowns probabilistically. That is, given historical data of a processor type, we have a probability distribution describing the likelihood of a given system slowdown occurring. Let $\boldsymbol{\mathcal{S}_j}$ be a random variable describing the slowdown of processor type j. A system slowdown of x will scale the execution time of the application by x.

3) Combining Uncertainties: Let $\boldsymbol{\mathcal{X}_{i,j,k}}$ be a random variable describing the final execution time distribution of application i running on k processors of type j. To obtain the *final execution time distribution*, multiply the *parallel execution time distribution* by the *system slowdown distribution* [10], $\mathcal{S}_j$, as defined by Eq. 1. Once $\mathcal{X}_{i,j,k}$ is obtained, it can be used to calculate the probability that a given resource allocation will result in application i finishing by a given time on k processors of type j. The multiplication of the distributions was performed discretely in the simulation code, not using a closed form solution (as one does not exist for the types of distributions used).

$$\mathcal{X}_{i,j,k} = \mathcal{S}_j \frac{\mathcal{T}_{i,j}}{S(k)} \tag{1}$$

D. Robustness

A "robust" resource allocation is defined as a resource allocation that mitigates the effect of uncertainties on a given performance objective. To claim robustness for a system, the following three questions must be answered [9]: **(1)** What performance feature makes the system robust? **(2)** What uncertainties is the system robust against? **(3)** How is robustness quantified?

To answer the robustness questions, the *makespan goal*, $\boldsymbol{\Delta}$, is defined as a target time for all applications to attempt to complete executing by, as well as a time used to calculate the probability that a given resource allocation will complete by a given Δ. Therefore, the performance feature that makes the system robust is applications completing by the makespan goal, Δ. The system is robust against uncertainties in application execution times and the uncertainties in system slowdown.

Let $\boldsymbol{P(i,(j,k))}$ be the probability that application i allocated resources (j,k) completes by the makespan goal Δ, obtained by evaluating the cumulative distribution function (**cdf**) of $\mathcal{X}_{i,j,k}$ at Δ. The robustness of the resource allocation I, denoted $\Psi(I)$, is quantified in Eq. 2.

$$\Psi(I) = \min_{i=1..N_a} P(i, I[i]) \tag{2}$$

E. Formal Problem Statement

We are given a batch of N_a moldable parallel applications to allocate resources to from N_p heterogeneous processor types, where processor type j has n_j processors. We know the parallel characteristics and serial execution time distribution for each application i on each processor type j. We also know the system slowdown distribution for each processor type j. Within the constraint of an allocation not exceeding the total number of processors of each type and the constraint that each application can only be assigned processors of one type, the goal of the resource allocation heuristics is to find a resource allocation I to maximize $\Psi(I)$, given in Eq. 3 as the performance objective $\boldsymbol{\rho}$.

$$\rho = \max \Psi(I) = \max \min_{i=1..N_a} P(i, I[i]) \tag{3}$$

III. Heuristics

A. Processor Balance

1) Overview: The goal of the *processor balance* heuristics is to give equal resources (i.e., processors) to each application. Let the total number of processors in the system be $\boldsymbol{\tau}$ (i.e., $\tau = \sum_{j=1}^{N_p} n_j$) and let $\boldsymbol{M}$ be the average number of processors per application in the system (i.e., $M = \frac{\tau}{N_a}$). We assume that M is an integer value and each n_j is a multiple of M for each processor type. These assumptions allow the comparison of our proposed robustness floor heuristics to the processor balance heuristics, but the assumptions are not necessary for the robustness floor heuristics.

The processor balance heuristics will then split the total number of resources into N_a blocks of M processors, where each block is comprised of processors of a single type. This reduces the dimensionality of the problem to just assigning an application to a processor type. The two variants of how this assignment is accomplished are *random* and *smart*. These two heuristics will be used as a comparison to the performance of the robustness floor heuristics, as a layperson might assign a fair share of resources to each user [16].

2) Random: The *processor balance – random* (**PB-R**) variant of the processor balance heuristic assigns the N_a applications to the N_a groups of processors randomly. That is, for each of the N_a groups of M processors, randomly select an application to be assigned. The application and processor group are removed from further allocation decisions and the process is repeated until no applications remain.

3) Smart: Unlike the PB-R variant, *processor balance – smart* (**PB-S**) greedily assigns applications to processor groups. Each of the N_a groups of M processors has an associated processor type j. Randomly select a group and assign the application i that maximizes $P(i,(j,M))$. The application and processor group are removed from further allocation decisions and the process is repeated until no applications remain.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

B. Robustness Floor

1) Overview: Our proposed robustness floor algorithm, shown as pseudocode in Fig. 3, is an iterative-greedy heuristic with three variants. For each algorithm iteration l, the robustness floor algorithm performs a binary search over the minimum value of robustness an application should achieve, $\boldsymbol{\Omega_l}$ (i.e., each application should have at least a probability of Ω_l of completing by the makespan goal). Let $\boldsymbol{\Pi(\Omega_l)}$ be a function that returns a matrix where the i, j element gives the minimum number of processors k required for application i on processor type j to meet $P(i,(j,k)) \geq \Omega_l$. Also let $\boldsymbol{greedy(\Pi(\Omega_l))}$ be a greedy heuristic that takes the application-processor pairings from $\Pi(\Omega_l)$ and returns a complete resource allocation I. These greedy heuristics are described in Subsections III-B2 to III-B4. After the greedy heuristic returns a resource allocation I, the function $\boldsymbol{v(I)}$ returns true or false depending on whether I is valid (i.e., every application is assigned a set of resources). If $\boldsymbol{\lambda_l}$ is the current binary search residual, defined as $\Omega_l - \Omega_{l-1}$, the next robustness floor value will be changed by $\frac{\lambda_l}{2}$. If $v(I)$ returns true, then the robustness floor of the next iteration $\Omega_l = \Omega_l + \frac{\lambda_l}{2}$, but if $v(I)$ returns false, $\Omega_l = \Omega_l - \frac{\lambda_l}{2}$. Finally, let $\boldsymbol{\Lambda}$ be the minimum residual considered (i.e., when $\lambda_l < \Lambda$, stop iterating).

1: $\Omega_l = 1.0$
2: $\Omega_{l-1} = 0.0$
3: **repeat**
4: $\quad I = greedy(\Pi(\Omega_l))$
5: $\quad \lambda_l = \Omega_l - \Omega_{l-1}$
6: $\quad \Omega_{l-1} = \Omega_l$
7: $\quad$ **if** $v(I) ==$ **true then** $\Omega_l = \Omega_l + \frac{\lambda_l}{2}$
8: $\quad$ **else if** $v(I) ==$ **false then** $\Omega_l = \Omega_l - \frac{\lambda_l}{2}$
9: **until** $\lambda_l < \Lambda$
10: **return** I

Fig. 3. Robustness floor algorithm

2) Min-Min Processors: The min-min processors greedy heuristic is a two-stage greedy heuristic (e.g., [20], [21]) that describes one of the $greedy$ functions from line 4 in Fig. 3. In the first stage, for each application i, find the processor type j in row i of the matrix returned by $\Pi(\Omega_l)$ that uses the minimum number of processors. In the second stage, from the application to processor type pairs found in the first stage, assign the application, $\boldsymbol{i_{min}}$, to the processor type, $\boldsymbol{j_{min}}$, that uses the minimum number of processors, $\boldsymbol{k_{min}}$. Remove application i_{min} from further allocation decisions and decrement the number of processors of type j_{min} by k_{min}. Repeat the two stages until all applications are assigned (i.e., $v(I) =$ true) or until there are not enough remaining processors to make any more allocations (i.e., $v(I) =$ false).

By utilizing the min-min processors greedy heuristic in step 4 of the robustness floor algorithm, the *robustness floor min-min* (**RF Min-Min**) heuristic is formed.

3) Min-Max Processors: The min-max processors heuristic is a two-stage greedy heuristic similar to the min-min processors heuristic in that the first stage is the same. In the second stage, however, from the application to processor type pairs found in the first stage, assign the application, $\boldsymbol{i_{max}}$, to the processor type, $\boldsymbol{j_{max}}$, that uses the maximum number of processors, $\boldsymbol{k_{max}}$. Remove application i_{max} from further allocation decisions and decrement the number of processors of type j_{max} by k_{max}. Repeat the two stages until all applications are assigned (i.e., $v(I) =$ true) or until there are not enough remaining processors to make any more allocations (i.e., $v(I) =$ false).

The intuition behind assigning those applications that need more processors first is that as more applications are assigned, it is harder to find room for those requiring more processors to meet the robustness floor. By utilizing the min-max processors greedy heuristic in step 4 of the robustness floor algorithm, the *robustness floor min-max* (**RF Min-Max**) heuristic is formed.

4) Duplex: The final greedy heuristic is a combination of the min-min and min-max processors heuristics, referred to as duplex. At step 4 in Fig. 3, duplex runs both min-min and min-max processors at each iteration and returns I such that the performance objective, ρ, is maximized. By utilizing the duplex greedy heuristic in step 4 of the robustness floor algorithm, the *robustness floor duplex* (**RF Duplex**) heuristic is formed.

IV. Simulation Parameters

A. Overview

The following section describes parameters that are used only to conduct a simulation study for analysis. The techniques introduced above can be used for any real system. Regardless of the input data, a system administrator utilizing these techniques would need to evaluate their effectiveness for their exact system. The performance from our simulation analysis does not imply the same performance on *all* systems.

B. Application Parameters

To model the stochastic execution times, Gamma distributions are used [12]. Gamma distributions were chosen to represent the application execution times as they are non-negative and their shape is flexible, allowing the representation of execution time distributions of a myriad of different applications. To generate the serial execution time distributions for each application on each processor type, the Coefficient of Variation (**COV**) method was used [22] to obtain the mean and standard deviation of each application on each processor type. To generate the parallel characteristics for the Downey model for each application, the distributions from [23] are used.

C. System Slowdown

To represent the slowdown of a given processor type, a modified form of a Beta distribution was used because it is a flexible distribution on the interval [0,1] that was similar to the shape of small scale slowdown studies we conducted on actual systems. It can be used to model the *system availability*, defined as the inverse of the system slowdown, where in this context 0 corresponds to no availability and a slowdown of ∞, 1 corresponds to a fully available system and no slowdown, and a processor type that is 50% available would have a slowdown of 2. To use the Beta distribution as a model for the system slowdown, the reciprocal of the x-axis is used (i.e., the x-axis is now on the interval $[1, \infty)$ instead of $[0, 1]$). We denote the *overall system slowdown* as $\boldsymbol{\Gamma}$, defined as the weighted average (weighted by the number

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

of processors for each type) of the mean system slowdown of the processor types.

D. Makespan Goal

Let $\boldsymbol{\mu_{i,j,k}}$ be the mean execution time of application i using k processors on processor type j. The calculation of the makespan goal, Δ, is described in Eq. 4.

$$\Delta = \frac{\sum_{i=1}^{N_a} \sum_{j=1}^{N_p} \mu_{i,j,M}}{N_a N_p} \tag{4}$$

The intuition behind using Eq. 4 is that it represents an average of the applications' performance in the given heterogeneous system. The inner summation averages the mean execution time of an application across all processor types. The outer summation averages across all applications. This allows different scenarios (e.g., number of applications, number of processor types) to be compared by using the same makespan goal calculation. We are aware that the determination of the makespan goal will have an impact on the resource allocation determined by the heuristics. This is currently out of the scope of this paper, but will be explored in the future.

E. Upper Bound on Robustness

Because robustness is defined across all applications as the minimum probability $P(i,(j,k))$ that an application i in resource allocation I completes by the makespan goal Δ, it is possible to find an upper bound on robustness based upon the worst performing application. The upper bound (given in Eq. 5), $\boldsymbol{B}$, states that for each application i, find the processor type j that maximizes its probability of completing by the makespan goal if it is given *all* processors of that type (i.e., n_j). Out of all of those probabilities, the application that has the minimum probability of completing by the makespan goal sets the upper bound on system robustness.

$$B = \min_{i=1..N_a} \max_{j=1..N_p} P(i,(j,n_j)) \tag{5}$$

V. Simulation Results

For our analysis, we consider three different batch sizes (i.e., N_a) of 8, 32, and 128 applications. The total number of processor types (i.e., N_p) explored for each batch size were $\{1,2,4,8\}$, $\{1,2,4,8,16\}$, and $\{1,2,4,8,16,32\}$, respectively. The total number of processors in the system (i.e., τ) with batch sizes of 8, 32, and 128 were 64, 256, and 1024, respectively, where n_j varies between types. The overall system slowdown, Γ, was broken into categories of low, mixed, and high corresponding to Γ of 1.1, 1.36, and 1.6, respectively. Each scenario (where a scenario is a batch size, system size, and system slowdown category) was run for 48 trials for each of the five heuristics with the 25th, median, and 75th quartiles shown. The plus symbols show all trials outside of the 25th and 75th quartiles. Between trials, the application characteristics and the makespan goal differed. The stopping criterion for the robustness floor heuristics, Λ, was set to 0.01. This led to each robustness floor variant running for seven iterations ($\lceil \log_2(\Lambda^{-1}) \rceil$).

A typical result is presented in Fig. 4. This figure shows the robustness compared between the five heuristics and the upper bound. The batch size was 32 applications with four processor types with high slowdown and 256 total processors.

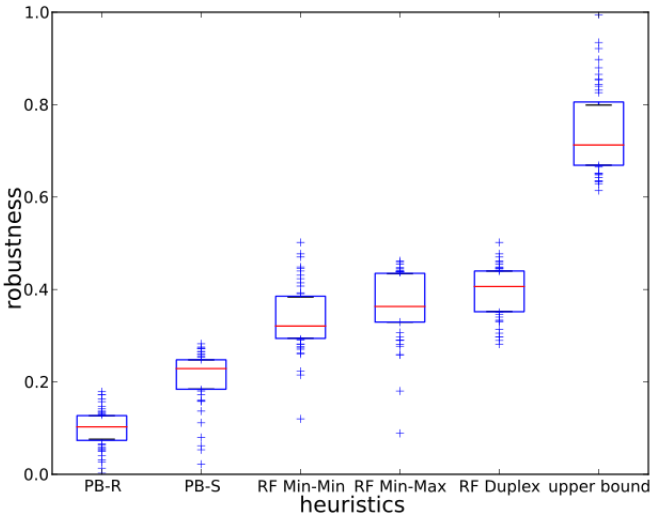

Fig. 4. A typical result in the comparison of the five heuristics and the upper bound. The batch size was 32 applications with four processor types with high system slowdown. The box plot shows the distribution of 48 trials with the 25th, median, and 75th quartile trials represented by the box. All trials outside of the 25th and 75th quartile are shown with the plus symbol.

We can see that the two processor balance heuristics do not perform as well as the three robustness floor heuristics. This is because the processor balance heuristics only make decisions on which processor type to allocate an application to, where the number of processors per application is fixed as M. The PB-S heuristic performs better than the PB-R heuristic because at each allocation decision it assigns the application that will have the highest probability of completing by the makespan goal where PB-R makes random allocation decisions. Out of the robustness floor heuristics, RF Min-Max, in general, performs better than RF Min-Min. This is because as more applications are allocated, there is less room and it becomes harder to assign the applications that need more processors. RF Min-Max assigns these larger applications first, leading to better performance in most cases. RF Duplex will always perform at least as well as RF Min-Min and RF Min-Max because it runs both greedy algorithms at each iteration. By using intelligent calculation optimizations, running this algorithm only requires 5 to 10% longer than either RF Min-Min and RF Min-Max. The upper bound is not overlapped by any of the heuristics because, in general, it is not achievable as it assumes an application occupies an entire processor type. This does not leave enough resources for the remaining applications to have an ample opportunity of completing by the makespan goal.

Because RF Duplex was the best performing heuristic in all scenarios with respect to robustness, we focus the rest of the discussion in regards to it. In our simulations, we noticed that the upper bound remained mostly constant for any given number of processor types. This is because the upper bound is only set by one application on one processor type, therefore the number of processor types does not matter in that calculation, but rather the heterogeneity and performance of a single processor type in the system. Where the number of processor types does matter, however, is with the heuristics. The more processor types there are, the more the heuristics can leverage the benefit of the heterogeneity in the system.

The heuristics leveraging the heterogeneity in the system is apparent in Fig. 5. Let $\boldsymbol{B_{norm}}$ be difference between the

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

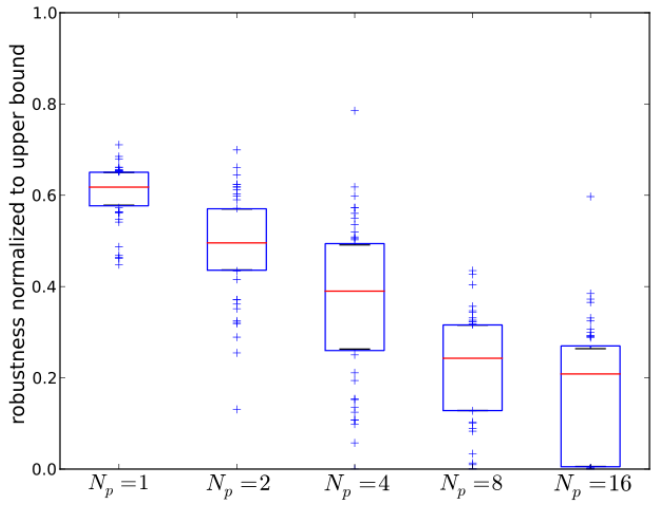

Fig. 5. The trend in performance, in terms of B_{norm}, when increasing the number of processor types in the system. A B_{norm} value of zero indicates that the robustness exactly equals the upper bound (i.e., $\Psi(I) = B$). The batch size was 32 applications and the system had a low system slowdown. The number of processor types varied between 1, 2, 4, 8, and 16.

upper bound and the robustness, normalized by the upper bound (i.e., $B_{norm} = \left(\frac{B-\Psi(I)}{B}\right)$). As B_{norm} approaches zero, the performance of the resource allocation approaches the upper bound. Fig. 5 shows how B_{norm} changes when the number of processor types are varied in a system with a batch size of 32 and low system slowdown. The increase in the number of processors in the system leads to better performance by RF Duplex with respect to the upper bound.

VI. Conclusions

A robustness metric was presented that combines the uncertainties of moldable parallel applications with stochastic execution times and heterogeneous resources with uncertain availability. Using knowledge of the parallel characteristics of the application in conjunction with the robustness metric, three iterative-greedy heuristics were designed and studied through simulation. In practice, the RF Duplex heuristic should be used. For a small increase in computation time, it combines the benefits of RF Min-Min and RF Min-Max.

In the future, additional resource allocation heuristics will be designed, implemented, and analyzed. The sensitivity of the performance of the heuristics to the setting of the makespan goal will be explored. Last, as in [10], we will combine the resource allocation techniques with a second stage that implements dynamic loop scheduling, a suite of runtime performance optimization techniques [24].

Acknowledgments

The authors thank M. Oxley and K. Tarplee of Colorado State University for their valuable comments.

References

[1] M. M. Eshaghian, *Heterogeneous Computing*. Artech House Publishers, 1996.

[2] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for heterogeneous computing systems," *Advances in Computers Volume 63: Parallel, Distributed and Pervasive Computing*, Ali R. Hurson, Ed. Elsevier, 2005.

[3] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 885–900, Aug. 2005.

[4] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *Journal of Parallel and Distributed Computing*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

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

[6] C. Boneti, R. Gioiosa, F. J. Cazorla, and M. Valero, "Using hardware resource allocation to balance HPC applications," *Parallel and Distributed Computing*, Ros Alberto, Ed. InTech, 2010.

[7] R. Wolski, N. Spring, and J. Hayes, "Predicting the CPU availability of time-shared Unix systems on the computational grid," *Cluster Computing*, vol. 3, no. 4, pp. 293–301, Dec. 2000.

[8] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[9] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems," *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Ed. Boca Raton, FL: Chapman and Hall/CRC Press, 2008.

[10] F. M. Ciorba, T. Hansen, S. Srivastava, I. Banicescu, A. A. Maciejewski, and H. J. Siegel, "A combined dual-stage framework for robust scheduling of scientific applications in heterogeneous environments with uncertain availability," in *21st Heterogeneity in Computing Workshop (HCW 2012) in the proceedings of the IEEE International Parallel and Distributed Processing Symposium*, May 2012, pp. 193–207.

[11] D. J. Robb and E. A. Silver, "Probability density functions of task-processing times for deterministic, time-varying processor efficiency," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1049–1052, Nov. 1990.

[12] S. R. Lawrence and E. C. Sewell, "Heuristic, optimal, static, and dynamic schedules when processing times are uncertain," *Journal of Operations Management*, vol. 15, no. 1, pp. 71–82, Feb. 1997.

[13] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, June 2007.

[14] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications," in *19th International Conference on Supercomputing*, June 2005, pp. 303–312.

[15] R. Ruiz and T. Stützle, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," *European Journal of Operational Research*, vol. 187, no. 3, pp. 1143–1159, June 2008.

[16] *Maui Scheduler^{TM} Administrator's Guide*, Adaptive Computing Enterprises, Inc. [Online]. Available: http://docs.adaptivecomputing.com/maui

[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

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

[19] A. B. Downey, "A parallel workload model and its implications for processor allocation," *Cluster Computing*, vol. 1, no. 1, pp. 133–145, May 1998.

[20] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hengsen, and R. F. Freund, "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.

[21] M. Maheswaran, S. Ali, H. J. Siegel, D. Hengsen, 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. 107–131, Nov. 1999.

[22] S. Ali, H. J. Siegel, M. Maheswaran, D. Hengsen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering,* Special Tamkang University 50th Anniversary Issue, vol. 3, no. 3, pp. 195–208, Nov. 2000. Invited.

[23] W. Cirne and F. Berman, "Using moldability to improve the performance of supercomputer jobs," *Journal of Parallel and Distributed Computing*, vol. 62, no. 10, pp. 1571–1601, Oct. 2002.

[24] I. Banicescu and R. L. Cariño, "Addressing the stochastic nature of scientific computations via dynamic loop scheduling," *Electronic Transactions on Numerical Analysis*, vol. 21, pp. 66–80, 2005.