2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

A Combined Dual-stage Framework for Robust Scheduling of Scientific Applications in Heterogeneous Environments with Uncertain Availability

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

*Technische Universität Dresden
Center for Information Services
and High Performance Computing
01062 Dresden, Germany
florina.ciorba@tu-dresden.de

†Mississippi State University
Department of Computer
Science and Engineering
Mississippi State, MS 39762, USA
{*ioana@cse.,ss878@*}*msstate.edu*

Colorado State University
‡Department of Electrical and
Computer Engineering
§Department of Computer Science
Fort Collins, CO 80523, USA
{*timothy.hansen,aam,hj*}*@colostate.edu*

***Abstract*—Scheduling parallel applications on existing or emerging computing platforms is challenging, and, among other attributes, must be efficient and *robust*. A dual-stage framework is proposed in this paper to evaluate the robustness of efficient resource allocation and dynamic load balancing of scientific applications in heterogeneous computing environments with uncertain availability. The first stage employs robust resource allocation heuristics, while the second stage incorporates robust dynamic loop scheduling techniques. The combined dual-stage framework constitutes a comprehensive framework that enables and provides guarantees for the robust execution of scientific applications in computing systems where uncertainty is caused by various unpredictable perturbations. The paper reports on studies for determining the best techniques to be used for each stage that: (a) maximize the probability that the system makespan satisfies a deadline, and (b) minimize the system makespan for every given availability level in the system. The usefulness and benefits of the proposed framework are demonstrated via a small scale example.**

***Keywords*-high performance; heterogeneous systems; non-dedicated systems; resource allocation; dynamic loop scheduling; robustness; uncertainties.**

I. Introduction

The rapid development of computing technology has increased the complexity of computational systems and their ability to solve large, and more complex, scientific problems. These computing systems often are heterogeneous and operate in uncertain environments, consisting of computing resources that differ in number and availability over time. Machine availability is the percentage of the machine's total computational resource that can be used for a given application. A machine is said to be loaded when its availability is less than 100%.

Scientific applications express the solutions to complex scientific problems, which often are data-parallel and contain large loops. The execution of such applications in heterogeneous computing environments is computationally intensive and exhibits an irregular behavior, in general due to variations of algorithmic and systemic nature [1, ch. 4]. Distribution of input data and variations of algorithmic nature cause intrinsic imbalance, while variations of systemic nature cause extrinsic imbalance [2]. Load imbalance in computationally intensive scientific applications is often their major performance degradation factor [1][2]. Traditionally, solutions that address load imbalance in scientific applications involve dynamic data and/or work re-distribution.

Our problem statement has two components. First, given a collection of applications with uncertain input data and a heterogeneous computing system with uncertain availability, how can resources be assigned to maximize the probability that applications can complete by a common deadline?

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.5

Second, given this allocation of resources, how can we minimize the makespan for this collection of applications?

The work presented herein demonstrates that using robust resource allocation (**RA**) heuristics [3] *and* application load balancing via dynamic loop scheduling (**DLS**) techniques, in concert, will enhance the execution of computationally intensive scientific applications in uncertain heterogeneous systems. The goal of this research is to assign applications to heterogeneous computing systems and execute them in such a way that all applications complete before a common deadline, and their completion times are robust against uncertainty in input data *and* system availability.

To accomplish this goal, the approach proposed herein is to divide the execution of scientific applications on heterogeneous computing systems into two stages, as outlined in Figure 1:
Stage I *initial mapping*–resources are allocated to each application according to a given robust RA policy.
Stage II *runtime application scheduling*–the execution of each application is optimized, for the set of resources allocated in the previous stage, according to a given robust application scheduling strategy.

Initial mapping (**IM**) can be defined as the problem of finding a mapping of a batch of applications onto a set of resources to maximize robustness against uncertain input data and system availability. Robustness here is defined as the probability that applications are completed on the allocated resources by a common deadline [4].

Motivation for Stage I. The motivation for solving the IM problem via robust RA is to avoid the runtime resource reallocation problem, i.e., reallocating resources already assigned to applications to avoid violations of the performance objective. The robustness of an RA can be quantified as the joint probability that all applications will complete by their deadline given the uncertain input data and system availability.

Motivation for Stage II. Just as in stage I, uncertain runtime availability of resources allocated to an application, as well as uncertain input data, are known sources of uncertainty in stage II and may impact the applications execution times. The motivation for this stage is based on the assumption that a specific runtime application scheduling (**RAS**) policy exists that avoids the runtime resource reallocation problem and that satisfies the stated performance objective, while possibly allowing a larger degree of uncertainty in input data and system availability.

RAS is defined as the problem of selecting the DLS technique for dynamic load balancing of applications during their execution on the resources already allocated in stage I that maximizes robustness against uncertain input data and system availability [5][6], defined as the maximum allowable decrease in the expected availability of the resources allocated in stage I before a performance objective violation occurs.

Employing a robust DLS technique for an application during stage II will allow the application to begin and complete its execution on the same set of resources that have been allocated during stage I, while in case of perturbations, only iterations (of the same application) will be migrated between the processors of the allocated resource set.

Usefulness. The usefulness of the proposed combined dual-stage framework is based on the following hypothesis: using an intelligent approach in both stages will result in better overall system performance than using an intelligent approach for either stage in isolation or neither. The dual-stage framework allows investigation of the overall degree of tolerable uncertainty, such that the desired performance objective is satisfied, for each application individually *and* the entire collection of applications running on the heterogeneous computing system.

Contribution. The main contribution of this paper is the design of an intelligent two-stage framework to solve the problem of allocating resources to applications to maximize the probability that the applications can complete by a common deadline given uncertainty in the input data and system availability, including developing a mathematical model of this environment.

The next section presents a review of related work, RA heuristics, and DLS techniques. The

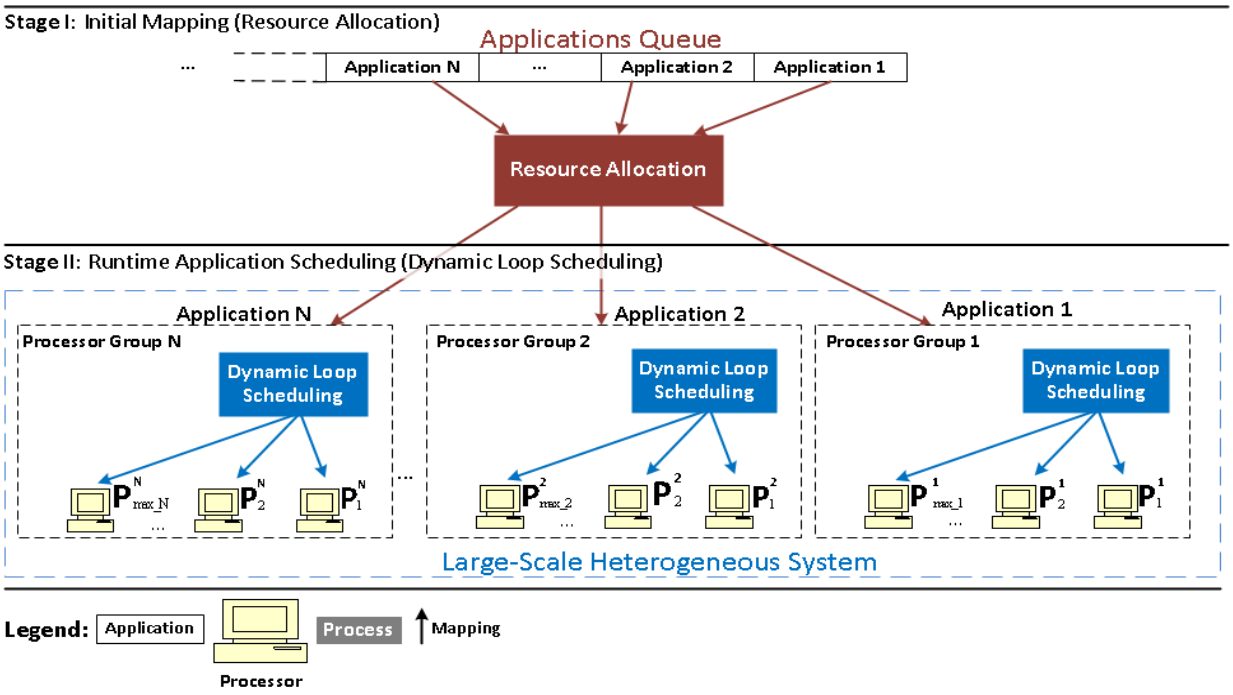

Figure 1: Schematic illustration of the proposed dual-stage framework. A RA heuristic is employed in stage I to assign each application from a batch of N applications in the queue to one of the N groups of processors of a large-scale heterogeneous system. DLS techniques are used in stage II for runtime scheduling of each application onto the processors of their respective assigned group.

proposed combined dual-stage framework is described in Section III. The usefulness and benefits of the proposed framework are demonstrated via a small scale example in Section IV. The conclusions and insights into future work are summarized in Section V.

II. Related Work

This work spans two major research areas: resource allocation and dynamic application scheduling in heterogeneous computing systems. The following is a brief summary of some work relevant to both research areas.

A framework for resource allocation and task scheduling is proposed in [7] for efficient allocation of heterogeneous grid resources to resource-intensive applications that minimizes their makespan and allocates the minimum number of resources. This approach is static and based on the current state of the grid resources. In contrast to this work, the resource allocation and application scheduling strategies are intertwined, application tasks are assumed to take one unit of time, and no source of uncertainty is considered.

The problem of mixed resource allocation and task scheduling has been addressed via a constrained-based approach as a temporally constrained and a resource constrained problem [8]. Time constraints can be limit times and precedences, while resource constraints concern allocation and sharing. Constraints propagation mechanisms have been proposed that led to the removing of some task assignments, or that determined inconsistent allocations between pairs of tasks. In contrast to this work, it is assumed that the durations of the unassigned tasks are correlated and constant, and that resources are homogeneous.

Extensions to application and performance models used in compile-time task and resource allocation have been proposed that capture applications with statistical variations in execution times and task dependencies [1]. In contrast to this work, the focus is on compile-time mapping of single applications onto homogeneous multiprocessor platforms.

The approach proposed herein to satisfy the stated performance objective in the presence of uncertainties is to divide the execution of scientific

applications into two stages: stage I employs a RA heuristic to allocate a set of resources to every application, while in stage II, DLS techniques are employed (one for each application) to ensure an effective application execution on the set of resources allocated in stage I. Existing work on RA and DLS is reviewed next.

A. Review of RA heuristics

In general, the resource allocation and the application scheduling problems are both known to be NP-Complete [9][10][11], which leads to the use of scheduling heuristics. In the stochastic resource allocation model, the historical computing time of each task to be run on each of the machines in the system is said to be known beforehand and is represented as a probability mass function (**PMF**) [4]. The use of PMFs allows for the calculation of the probability of a machine finishing its tasks before a specified time. Because each of the machine's execution times are independent, the overall probability of the system completing by a specified time can be obtained by multiplying their probabilities together.

Due to the fact that the example given later in the paper (in Section IV) is illustrative and represents a small scale case, no particular RA heuristic is actually needed, as the optimal allocation can be determined exhaustively. As a comparison metric between possible resource allocations, a simple load balancing technique is used, in which each application is allocated an equal number of resources.

B. Review of DLS techniques

The most efficient dynamic load balancing approach for improving the performance of scientific applications employs DLS. This approach is effective in scientific applications that contain computationally intensive parallel loops. The DLS techniques are based on probabilistic analyses and ensure a high performance execution of the applications. Using DLS, a new size for the next chunk of ready-to-be-executed loop iterations is computed at runtime, and thereupon offered for execution to the first processor that finished executing other assigned chunks.

DLS methods provide two alternative approaches, *non-adaptive* and *adaptive*, for achieving good load balancing on variably loaded resources, as well as for executing iterations with varying execution times. Most of the techniques described in [12] are based on probabilistic analyses and are non-adaptive. Other non-adaptive techniques, not contained in the above survey, include fractiling and weighted factoring (**WF**) [13]. Subsequent efforts resulted in more elaborate techniques, called adaptive. A few examples include adaptive weighted factoring (**AWF**), and its variants: AWF-batch (**AWF-B**) and AWF-chunk (**AWF-C**), and adaptive factoring (**AF**) [14]. Most of these methods are derived from factoring (**FAC**) [15], and hence, employ rules similar to those of FAC to achieve dynamic load balancing.

The above adaptive methods are also based on probabilistic analyses. Their goal is to achieve the best possible scheduling that optimizes application performance (minimizing the makespan) via dynamic load balancing. The adaptive DLS techniques use a combination of *runtime* information about the application (e.g., input data) and the system (e.g., availability) to (i) predict the system capabilities for the next computational assignments, or (ii) to estimate the time the remaining application iterations will require to finish execution. These techniques dynamically compute the size of chunks (a collection of iterations) at runtime, such that they are completed within their optimal time with high probability.

The DLS techniques considered in stage II in this work are two non-adaptive methods, FAC and WF, and two adaptive methods, AWF-B and AF. The usefulness of the proposed dual-stage framework is not limited to this choice of DLS techniques. Due to space limitations, the interested reader is referred to the appropriate references for further details of the above DLS techniques.

III. A Combined Dual-stage Framework

A. Uncertainty and performance objective

A novel combined dual-stage framework (**CDSF**) is proposed herein, with the goal of assigning applications to heterogeneous computing

systems and executing them in such a way that all applications complete before a common deadline, and their completion times are robust against uncertain input data *and* system availability. The robust execution involves two stages: *initial mapping* using robust RA heuristics, in stage I, and *runtime application scheduling* using robust DLS techniques, in stage II. The CDSF provides a certain level of guarantee for satisfying the stated performance objective against uncertainties.

Uncertainty: The uncertainty is assumed to be caused by a *2-tuple* of perturbation parameters [3] (π_1, π_2), in which π_1 pertains to stage I, while π_2 pertains to stage II.

The execution time of each application is considered stochastic due to its dependence on input data. More specifically, we model the execution time of each application as a random variable and assume that we are given a PMF describing the possible execution time values and their probabilities for each combination of application and processor type. That is, the execution time of each application i when executed alone on a single, unloaded (fully dedicated) processor of type j is modeled as a random variable. The list of applications that the user may select from is assumed limited to a set of frequently requested algorithms such as may be found in companies or government research lab environments [4]. Consequently, we assume that the execution time random variable for each application is well characterized. That is, we assume that a PMF is available for each application execution time random variable on each processor type (determined by historical, experimental, or analytical techniques [16][17]). In addition, each application execution time is assumed independent, i.e., there is no inter-application communication. Similarly, the system availability for each processor type j is modeled as a random variable, $\alpha_\mathbf{j}$, with a given PMF describing the possible system availability percentages and their probabilities, generated using historical usage data of the heterogeneous computing system.

Let $\hat{\epsilon}$ be a matrix where the $(i,j)^{th}$ element is a random variable modeling the execution time for application i on processor type j, as described above. $\hat{\mathbf{A}}$ is a vector where the j^{th} element is α_j, denoting the availability of processors of type j, also described above. The perturbation parameter for stage I, π_1, is given by $\pi_1 = \{\hat{\epsilon}, \hat{\mathbf{A}}\}$. If a given application is executed on a single processor of a given type, its computation can be modeled based on $\hat{\epsilon}$ and $\hat{\mathbf{A}}$. However, because each application will be executed using parallelism, its computation time is more complex, and its modeling is described in Section IV.

Let $\mathbf{\Lambda}$ be the system load fluctuation at runtime [5][6][13]. Given $\mathbf{A} = 1 - \mathbf{\Lambda}$ as the *runtime* system availability in stage II, the perturbation parameter for stage II is $\pi_2 = \{\mathbf{A}\}$. In general, the runtime system availability can be higher or lower than the expected system availability, i.e., $\mathbf{A} \neq \mathbf{E}[\hat{\mathbf{A}}]$. A system is said to be loaded when it is less than 100% available. For example, a system having a load of 30% during a certain period of time, is said to have a $100\% - 30\% = 70\%$ availability for that period of time.

The uncertainty in this work is assumed to be caused by the 2-tuple $(\pi_1, \pi_2) = (\{\hat{\epsilon}, \hat{\mathbf{A}}\}, \{\mathbf{A}\})$.

Performance objective: The performance objective has two components, called performance features [3]. These performance features are expressed as a *2-tuple*, denoted (ϕ_1, ϕ_2), in which ϕ_1 is the performance feature related to stage I, and ϕ_2 is the performance feature of interest in stage II.

Let $\mathbf{\Psi}$ be the system makespan, defined as the completion time for an entire collection (or batch) of applications, determined by the maximum of the actual finishing times of all machines for all applications. $\mathbf{\Psi}$ represents the time when the next batch of applications will require resources given an RA heuristic used in stage I and a set of DLS techniques (one for each application) used in stage II. Let $\mathbf{\Delta}$ be the system deadline. Then ϕ_1 is defined as $\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta})$ given $\pi_1 = \{\hat{\epsilon}, \hat{\mathbf{A}}\}$, and $\phi_2 = \{\mathbf{\Psi}\}$ given $\pi_2 = \{\mathbf{A}\}$.

Given a batch of parallel scientific applications executing on the resources of a heterogeneous system, the performance objective in this work is given by the 2-tuple $(\phi_1, \phi_2) = (\{\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta})\}, \{\mathbf{\Psi}\})$, and is described as: (1) maximize the probability that all applications complete before

the system deadline, i.e., `maximize` ϕ_1 given $\hat{\epsilon}$ and $\hat{\mathbf{A}}$ (Stage I); and (2) minimize the system makespan that satisfies the deadline for every given availability in the system, i.e., `minimize` ϕ_2 given $\mathbf{A}$ (Stage II).

B. Outline of the proposed framework

The proposed CDSF for robust execution of scientific applications on heterogeneous uncertain computing systems is schematically illustrated in Figure 2.

Initial mapping conducted in stage I is the problem of finding a static mapping (i.e., one found in an offline planning phase) of a batch of applications onto a set of resources to maximize robustness of the allocation against uncertain input data and system availability, by maximizing the probability that all applications will complete before the deadline, given a PMF for system availability $\hat{\mathbf{A}}$. Runtime application scheduling carried out in stage II is the problem of finding a dynamic scheduling policy for each application that minimizes the parallel time to complete of an application for every given runtime system availability $\mathbf{A}$.

Stage I – initial mapping

Scientific applications arrive at random intervals in the queue of a resource manager, in view of assignment for execution onto any one of a group of resources of a heterogeneous computing system. The applications queue consists of different scientific applications, which can represent different instances of the same application.

As the applications arrive, their assignment to available resources is made in batches. After assignment, an application is placed in the input queue of the resource designated as coordinator (master) of the assigned group of resources. Any required data are staged at the master, in advance of application execution. Let N be the number of applications in the batch. Each application is assumed to be *data parallel* (with no interprocessor communications needed) and to contain large computationally intensive parallel loops.

Robust heuristics are employed for the initial mapping, and the intention is to conduct studies to determine the best heuristic to use in this stage. The best heuristic will provide the most robust mapping of groups of resources to applications, i.e., maximize the probability that an application completes before $\mathbf{\Delta}$, assuming a certain system availability $\hat{\mathbf{A}}$.

The resource allocation actions are pre-planned before the actual execution of the applications begins and the goal is to minimize (or to prevent) the immediate effects of uncertain perturbation in $\hat{\epsilon}$ and $\hat{\mathbf{A}}$ on the system makespan $\mathbf{\Psi}$, such that $\phi_1 = \{\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta})\}$ is maximized. Regardless of the type of allocated resources, once an allocation decision has been made, it cannot be adjusted for a currently executing application. Perturbations during the actual execution of applications are expected and addressed (or compensated for) in stage II via the use of robust DLS techniques.

Let max_i, $i = 1, N$ be the number of resources allocated to application i, and $T^{exp}_{max_i,i}$ be the expected time to complete of application i on max_i processors.

Stage II – runtime application scheduling

Each application from the current batch of N applications is executed on its group of resources allocated in stage I. A robust DLS technique from the set {FAC, WF, AWF-B, AF} [5][6][13] is employed to define the rules for executing an application at runtime. The intention is to conduct studies to determine the best DLS technique to employ for each application in the batch, such that the completion time of an application is minimized for every given runtime system availability $\mathbf{A}$, and consequently, the system makespan is smaller than or equal to the deadline. A single DLS technique may be employed for several applications as several distinct instances of the particular DLS technique. In general, the runtime system availability is expected to be different than the estimated system availability. In this work, it is assumed that $\mathbf{A} \leq \mathbf{E}[\hat{\mathbf{A}}]$.

The most robust DLS technique will provide the best runtime scheduling decisions for executing an application on the allocated group of processors that minimize the system makespan while tolerating a larger degree of perturbation in system availability than the one assumed in stage I. The

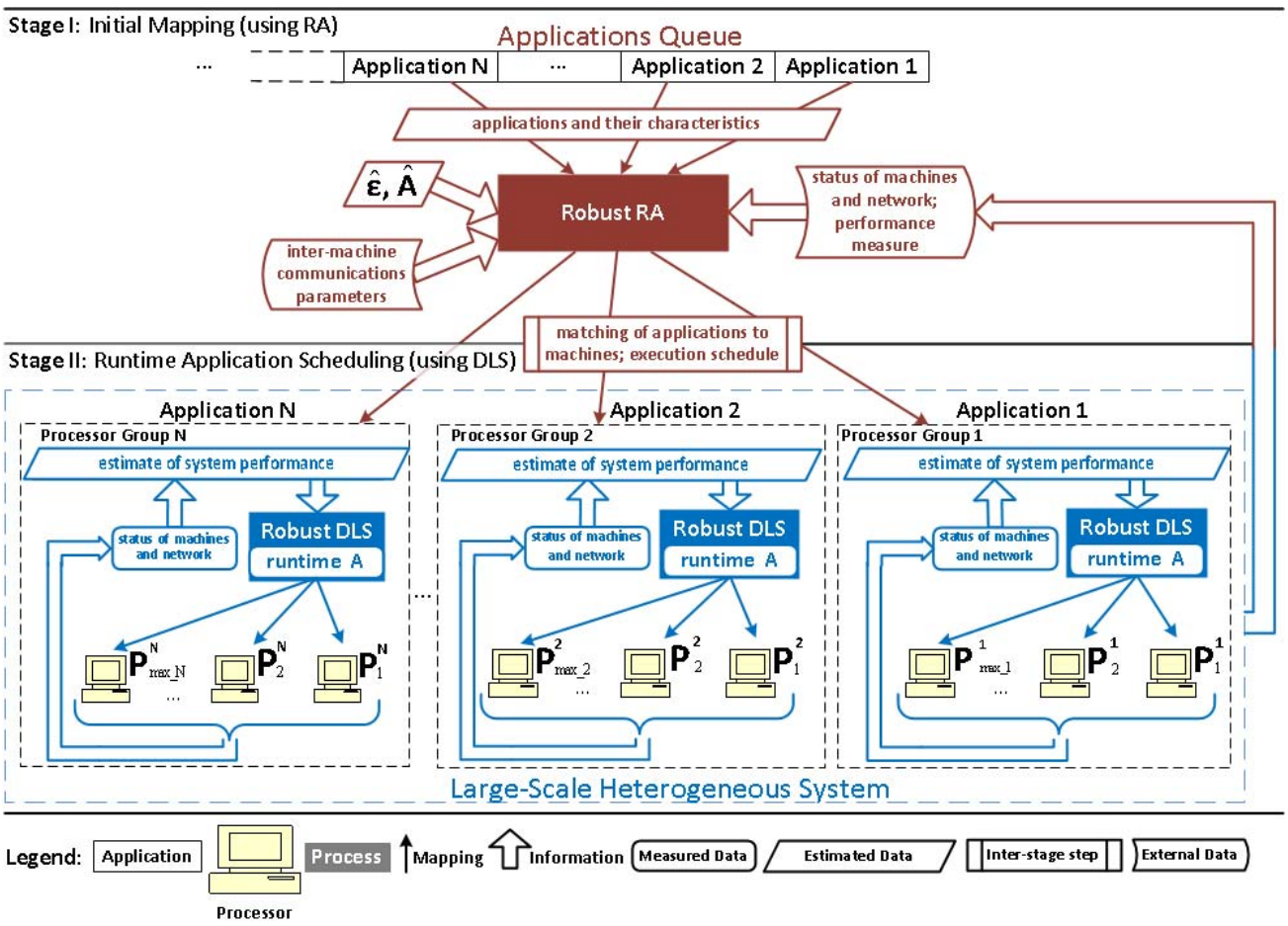

Figure 2: Schematic illustration of the proposed combined dual-stage framework: a robust RA heuristic is employed in stage I, and robust DLS techniques are employed in stage II. A number of N applications are mapped onto N groups of processors, which compose into a large-scale heterogeneous system with $\sum_{i=1}^{N} max_i$ processors.

goal of the robust DLS technique is to detect any runtime perturbation in system availability as soon as it occurs, and to take appropriate scheduling decisions for the remaining unexecuted application iterations. Stage II can, thus, be considered a runtime approach for the detection and recovery from the uncertain effects of the perturbation expected to occur in $\mathbf{A}$, on the performance feature ϕ_2 described earlier.

To guide the scheduling decisions at runtime and to tune the performance of an application, the DLS techniques use runtime estimations of the time required to compute loop iterations. These times are determined using probabilistic analyses and are influenced by the application input data and the availability to compute of the resource executing the iteration(s). The execution of an application using a DLS technique is non-preemptive, and, therefore, the choice of the DLS technique cannot be changed during runtime.

The overhead associated with employing a robust DLS technique is higher than that of a robust RA heuristic. The actions are not pre-planned and are taken dynamically during the application execution, as soon as perturbation occurs. The benefits are expected to, and in general do, compensate the overhead of employing robust DLS techniques.

C. Questions regarding the CDSF robustness

To claim robustness for a system, the following questions must be answered [18]:

1. What behavior of the system makes it robust?
Answer: The system considered in this work is robust if all applications complete before a common deadline $\mathbf{\Delta}$, given uncertainty in input data (which

impacts application execution time) and system availability. The system robustness is achieved via the CDSF employing robust RA and robust DLS in two consecutive stages.

A robust RA heuristic is one that is capable of maximizing the probability that all applications complete before the deadline. A DLS technique is said to be robust if it facilitated the execution of an application in the smallest amount of time, *and* if this time satisfies the deadline when the runtime system availability may vary from the one assumed initially.

2. What uncertainties is the system robust against?

Answer: Given uncertain variations in input data and system availability, application execution times are a known source of uncertainty in the system, and may have a significant impact on the stated performance objective. The uncertainty against which the system considered in this work is assumed to be robust is the 2-tuple (π_1, π_2).

3. How is the system robustness quantified?

Answer: The robustness of the system using the CDSF can be quantified as the joint robustness of the initial mapping in stage I and the runtime application scheduling in stage II. The robustness of stage I is quantified as the joint probability of all applications completing by the common deadline, i.e., ϕ_1. Let $\mathbf{A_i}$ be the PMF for a given case study. The robustness of stage II is quantified as the percent decrease in weighted system availability that can be tolerated by *all* applications without violating the deadline, i.e., $1 - (\mathbf{E}[\mathbf{A_i}]/\mathbf{E}[\hat{\mathbf{A}}])$ for which $\mathbf{\Psi} \leq \mathbf{\Delta}$.

Let ρ_1 be the largest robustness value of stage I. Also, let ρ_2 be the largest robustness value of stage II. The system robustness is quantified as the 2-tuple (ρ_1, ρ_2).

IV. Usefulness of Proposed Framework

The assessment of the usefulness of the proposed CDSF requires an investigation of the impact of the different possible RA heuristics and DLS techniques on the performance objective of interest. A small scale example is provided next to illustrate the usefulness of the proposed CDSF. The data that was chosen for this example was used to demonstrate the efficacy of the CDSF. The relevant assumptions for this example are described below.

System setup: Consider a heterogeneous system with twelve processors of two types, i.e., $j = 1, 2$: four processors of type 1, and eight processors of type 2. Processors of type 1 are assumed to have a different computational capacity and availability than processors of type 2. Case 1 in Table I describes the system availability as it was historically collected and aggregated over a given period of time, to form the expected system availability $\hat{\mathbf{A}}$ used in stage I, and is taken as a reference case. Cases 2–4 correspond to systems with decreased weighted availability compared to the reference case, i.e., $\mathbf{E}[\mathbf{A_1}] > \mathbf{E}[\mathbf{A_2}] > \mathbf{E}[\mathbf{A_3}] > \mathbf{E}[\mathbf{A_4}]$. Let p_j be the number of processors of type j, and e_j be the expected availability of processor type j. The weighted system availability can be calculated as shown in Equation 1.

$$\frac{\sum_{j=1}^{2}(p_j)(e_j)}{\sum_{i=1}^{3} max_i} \tag{1}$$

A batch of $N = 3$ applications is considered, having different sizes and serial/parallel component ratios (see Table II). Serial iterations can only be executed on a single processor and parallel iterations can be executed on multiple processors of the same type. The system deadline is $\mathbf{\Delta} = 3,250$ time units, and was chosen to help illustrate the differences between using intelligent stages versus naïve stages in the dual-stage framework.

The single processor execution times for each of the three applications on each of the two processor types are represented as PMFs; this is the $\hat{\epsilon}$ used in stage I. For this study, the PMFs were generated by sampling a normal distribution with the mean values (μ) shown in Table III. Each normal distribution used a standard deviation (σ) of one-tenth of its mean value, i.e., $\sigma = \frac{1}{10}\mu$. These values were considered to be the expected serial times required for each application to execute on one processor of a given type and were chosen to

highlight the usefulness of the proposed combined two-stage framework.

It is assumed that all applications must be assigned and that they are assigned to a power-of-2 number of processors of one type. For application i on processor type j, let T_{ijx} be the time of pulse x in the PMF, s_{ij} and p_{ij} be the serial and parallel fractions, respectively, and n_{ij} be the number of processors. Let T_{ijxn} be the pulse in the parallel execution time PMF of application i assigned to n_{ij} processors of type j. This specific parallel execution time PMF is obtained by recalculating each pulse of the single processor execution time PMF according to Equation (2).

$$T_{ijxn} = (s_{ij}T_{ijx}) + (p_{ij}T_{ijx})/n_{ij} \qquad (2)$$

For each pulse x, the time associated with T_{ijxn} will differ from T_{ijx}, while the probability will remain the same.

Table I: Processor availabilities by type and weighted system availabilities. Case 1 corresponds to $\hat{\mathbf{A}}$. Square brackets indicate $1 - (\mathbf{E}[\mathbf{A_i}]/\mathbf{E}[\hat{\mathbf{A}}])$.

	Proc.	Availability (%)	Probability (%)	Expected availability (%)	Weighted system availability (%)
Case 1 ($\mathbf{A}_1 = \hat{\mathbf{A}}$)	Type 1	75	50	87.50	75.00
		100	50		
	Type 2	25	25	68.75	
		50	25		
		100	50		
Case 2 ($\mathbf{A}_2$)	Type 1	50	90	52.50	53.87 [28.17]
		75	10		
	Type 2	33	45	54.55	
		66	45		
		100	10		
Case 3 ($\mathbf{A}_3$)	Type 1	52	50	60.58	51.92 [30.77]
		69	50		
	Type 2	17	25	47.60	
		35	25		
		69	50		
Case 4 ($\mathbf{A}_4$)	Type 1	33	75	41.25	50.42 [32.77]
		66	25		
	Type 2	20	50	55.00	
		80	25		
		100	25		

Table II: Characteristics of a batch of applications

App.	# Serial iterations	# Parallel iterations	% Serial iterations	% Parallel iterations
1	439	1024	30	70
2	512	2048	20	80
3	216	4096	5	95

Table III: Normal distribution mean values for single processor execution times of each application on each processor type.

Processor	$T^{exp}_{1,1}$	$T^{exp}_{1,2}$	$T^{exp}_{1,3}$
Type 1	1,800	2,800	12,000
Type 2	4,000	6,000	8,000

Once the PMF modeling the parallel execution time of an application on a certain number of processors of one type is calculated, it is convoluted with the PMF modeling the historical system availability ($\hat{\mathbf{A}}$) of processors of that type (α_j), to determine the PMF used in stage I to calculate the resource allocation robustness values. To calculate the probability that for a given resource allocation, an application will meet the common deadline $\mathbf{\Delta}$, each pulse in this resulting PMF corresponding to a time *less* than the deadline is summed together. Due to the fact that each application's finishing times are independent, the probability that the entire system will complete by the common deadline is given by multiplying each application's probability of completion by $\mathbf{\Delta}$ together.

To demonstrate the benefits and usefulness of the proposed CDSF for allocating the twelve heterogeneous processors of two types, executing with uncertainties shown in Table I, to the three applications, four scenarios have been identified: 1) naïve IM−naïve RAS, 2) *robust* IM−naïve RAS, 3) naïve IM−*robust* RAS, and 4) *robust* IM−*robust* RAS.

In all scenarios, the IM problem is solved assuming a system availability equal to $\hat{\mathbf{A}} = \mathbf{A}_1$, while the RAS problem is solved assuming the runtime system availability $\mathbf{A}$, is one value from the set $\{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4\}$.

In naïve IM, a simple load balancing technique is used to allocate an equal share of the available processors to each application. The load balancing allocation with the highest probability that all applications will complete before the deadline was chosen. In the system as described by the example, the load balancing technique allocated resources as described in Table IV. Given this resource allocation, the system has a 26% chance

of completing the applications before the deadline, i.e., $\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta}) = 26\%$.

Table IV: Resource allocation for naïve and robust IM

RA	App. i	Proc. type j	# Procs max_i
naïve IM	1	2	4
	2	1	4
	3	2	4
robust IM	1	1	2
	2	1	2
	3	2	8

In the robust IM case, all possible resource allocations are compared and the one with the highest probability of all applications completing before the system deadline is chosen. This results in resources being allocated as shown in Table IV. Given this resource allocation, the system has a 74.5% chance of completing the applications before the deadline, i.e., $\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta}) = 74.5\%$. It is important to note that this exhaustive search for the robust resource allocation is only feasible in the case of the small demonstrative example. More advanced and scalable RA heuristics are required for larger problem sizes, and our future work will include designing such robust RA heuristics.

Given the two resource allocations, naïve and robust, each application's expected completion time calculated in stage I is shown in Table V. These values were obtained by taking the expected value of the PMF relating to the assigned resources for each application. It is interesting to note that $T^{exp}_{max_2,2}$ is larger in the robust IM than in the naïve IM.

Table V: Parallel PMF estimated values of applications completion times for naïve and robust IM (in time units)

RA	$T^{exp}_{max_1,1}$	$T^{exp}_{max_2,2}$	$T^{exp}_{max_3,3}$
naïve IM	3800.02	1306.39	4599.76
robust IM	1365.46	1959.59	2699.86

In naïve RAS, straightforward parallelization is employed for each application to schedule its iterations in equal shares, which are then assigned to processors in a single step. This technique is called STATIC.

In robust RAS, a DLS technique from the set {FAC, WF, AWF-B, AF} is employed to execute an application on its allocated resources. The DLS techniques implement dynamic load balancing mechanisms based on probabilistic analyses to ensure minimal impact of runtime uncertainties on the application performance. For a given application and a runtime system availability, the DLS technique resulting in the smallest parallel execution time that satisfies the system deadline is considered best.

The usefulness of the proposed CDSF is based on the hypothesis that any of the first three scenarios will result in solutions that tolerate much *less* perturbations variations in the overall system, and therefore, are *less* robust. Thus, the fourth scenario (robust IM−robust RAS) is expected to be superior to the first three scenarios. The CDSF allows investigation of the overall degree of tolerable uncertainty for which the stated performance objective is satisfied, at the level of each individual application and for the entire batch of applications executing on the heterogeneous system.

Scenario 1) Naïve IM−naïve RAS

In this scenario, each stage employs naïve heuristics to allocate resources to each of the three applications, namely simple load balancing and STATIC, respectively. The application execution times are shown in Figure 3.

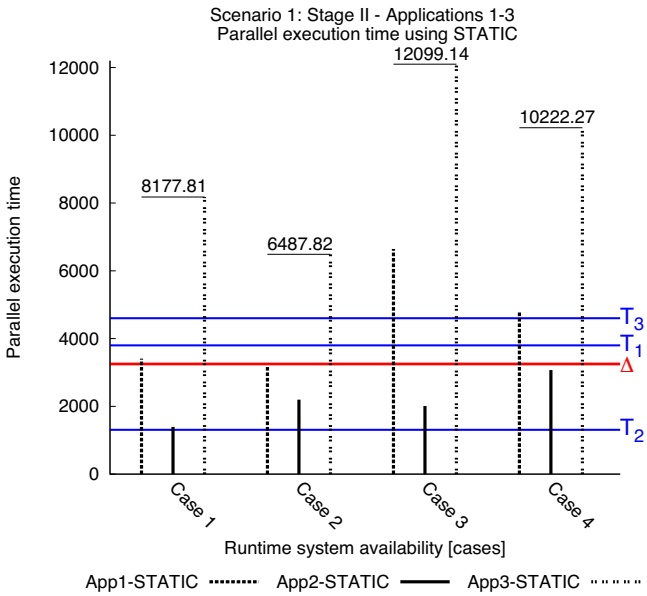

Figure 3: Scenario 1) Stage I: resource allocation using simple load balancing, Stage II: straightforward parallelization using STATIC. $\mathbf{\Delta} = 3,250$ time units is the system deadline. $T_1 = 3,800.02$ time units, $T_2 = 1,306.39$ time units, and $T_3 = 4,599.76$ time units where $T_i = T^{exp}_{max_i,i}$ (see Table V).

For the given resource allocation and application scheduling, $\phi_1 = 26\%$ and $\phi_2 > \boldsymbol{\Delta}$ for all system availability cases. This scenario shows that a naive RA policy in stage I and a straightforward parallelization in stage II cannot prevent the violation of the system deadline given the system availability cases considered. Therefore, the system is not robust.

Scenario 2) Robust IM−naïve RAS

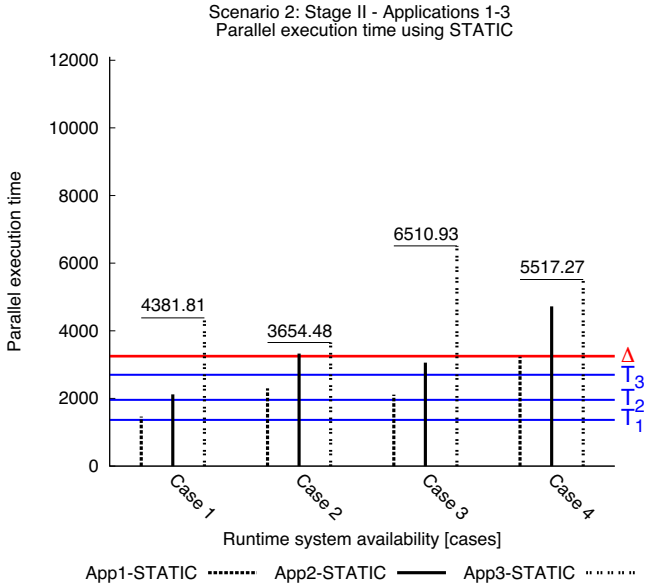

Figure 4: Scenario 2) Stage I: resource allocation using optimal RA, Stage II: straightforward parallelization using STATIC. $\boldsymbol{\Delta} = 3,250$ time units is the system deadline. $T_1 = 1,365.46$ time units, $T_2 = 1,959.59$ time units, and $T_3 = 2,699.86$ time units where $T_i = T^{exp}_{max_i,i}$ (see Table V).

The interest in this scenario is to investigate how much perturbation can be tolerated when only the robust IM ensures a certain level of robustness for all applications. Thus, an optimal RA heuristic is employed in stage I to allocate resources to each application, such that they all complete before $\boldsymbol{\Delta}$, with a probability of 74.5% assuming the system availability equals $\hat{\mathbf{A}}$ (see Table I). The use of robust RA is recommended when the assumptions made using a naïve RA are not sufficient to guarantee the satisfaction of $\boldsymbol{\Delta}$, as is the case in the previous scenario. Each application is parallelized using STATIC in stage II, and their execution times are plotted in Figure 4.

Given the robust resource allocation, $\phi_1 = 74.5\%$. This means that the system makespan has a higher probability of meeting the deadline when $\pi_2 = \mathbf{E}[\mathbf{A_1}]$ than in the previous scenario. However, Figure 4 shows that the performance of each application using STATIC degrades with decreasing system availability *after* the RA decisions have been taken in stage I, and $\phi_2 > \boldsymbol{\Delta}$ for all four system availability cases. Thus, it can be stated that the system is not robust.

Scenario 3) Naïve IM−robust RAS

In this scenario, the interest is to investigate how much perturbation can be tolerated when only the DLS policy ensures a certain level of robustness for each application. Thus, the allocation decisions are made in stage I according to a naïve RA heuristic. A robust DLS technique, i.e., FAC, WF, AWF-B, or AF, is employed in stage II, and uses knowledge obtained during the execution of the application about the system, to guide the scheduling decisions in such a way that the performance of the application suffers a minimal degradation with varying (decreasing) system availability. The application execution times for this scenario are illustrated in Figure 5.

For the naïve resource allocation the probability of all applications completing before $\boldsymbol{\Delta}$ is $\phi_1 = 26\%$. Even when the most robust DLS technique is used in stage II, one can note that certain applications finish earlier than others. This causes the violation of the system deadline, as it is the case for application 3 in case 1, and applications 1 and 3 in cases 2-4. Given that $\phi_2 > \boldsymbol{\Delta}$ for all four system availability cases, a more robust RA heuristic is, hence, required in stage I to complement the robust DLS technique used in stage II. It can be stated that the system in this scenario is not robust.

Scenario 4) Robust IM−robust RAS

The previous two scenarios show an improvement over the first scenario. This improvement is, however, insufficient to ensure that the system deadline is met for all applications. Therefore, in this scenario the largest amount of tolerable perturbation in system availability is considered, while ensuring that the system deadline is met for the *entire* batch of applications. This scenario, is

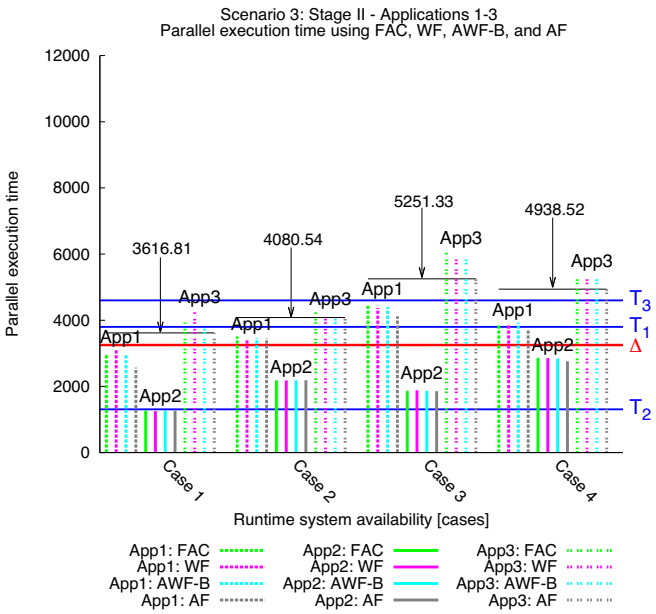

Figure 5: Scenario 3) Stage I: resource allocation using simple load balancing, Stage II: robust DLS using FAC, WF, AWF-B, and AF. $\boldsymbol{\Delta} = 3,250$ time units is the system deadline. $T_1 = 3,800.02$ time units, $T_2 = 1,306.39$ time units, and $T_3 = 4,599.76$ time units where $T_i = T^{exp}_{max_i,i}$ (see Table V).

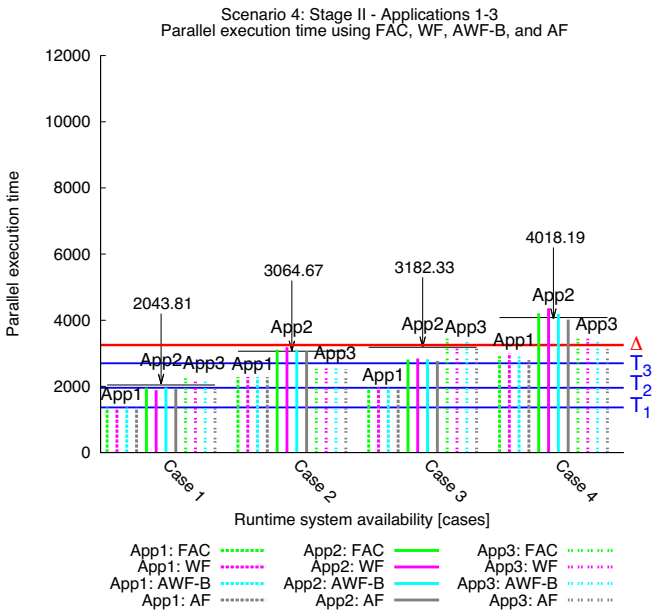

Figure 6: Scenario 4) Stage I: resource allocation using optimal RA, Stage II: robust DLS using FAC, WF, AWF-B, and AF. $\boldsymbol{\Delta} = 3,250$ time units is the system deadline. $T_1 = 1,365.46$ time units, $T_2 = 1,959.59$ time units, and $T_3 = 2,699.86$ time units where $T_i = T^{exp}_{max_i,i}$ (see Table V).

also referred to as the scenario that best illustrates the usefulness of the proposed combined dual-stage framework.

The robust IM from the second scenario (robust IM−naïve RAS) is employed in stage I to allocate a more suitable set of resources to each application than in scenarios 1 and 3. Just as in scenario 3, robust DLS algorithms are employed in stage II to minimize the impact on the application performance assuming unforeseen variation in the system availability, and to support the probability given by the robust IM in stage I that all applications complete before the system deadline.

The application execution times for this scenario are shown in Figure 6. One can note that the system deadline is met for all applications when the weighted system availability decreased by 28.17% (case 2) and 30.77% (case 3), respectively, compared to that assumed in stage I (case 1). When the weighted system availability decreased by 32.77% (case 4), the deadline is met for application 1, while it is violated for application 2 using any DLS technique and for application 3 using FAC, WF or AWF-B. This indicates that in case 4 and for application 3, AF is more robust than FAC, WF, or AWF-B when executing on the resources of type 2 allocated in stage I, with an expected availability of 55% for this processor type. Therefore, the system is said to be robust for system availability cases 1-3, while it is not robust for case 4, and the robustness value for stage I is $\rho_1 = 74.5\%$.

It is important to note that the above observations are valid only for the combination of type 1 and type 2 processors availabilities shown in Table I (third column), and not for any general combination that may result in the weighted system availability values in Table I (sixth column).

Table VI: Scenario 4) DLS techniques providing best application performance and meeting the system deadline for all cases of system availability

Application	Case 1	Case 2	**Case 3**	Case 4
1	WF	AF	**AF**	AF
2	WF	WF	**AF**	–
3	AF	AF	**AF**	AF

Table VI shows the DLS techniques that result in the best application performance and that at

the same time satisfy the system deadline. It can be noted that from all the cases considered in Table I the largest tolerable decrease in overall system availability compared to the reference case is 30.77% (case 3) and the system deadline is met for all applications. The most robust DLS technique in this case is AF for all applications (cf. Table VI, fourth column), and, hence, the robustness value for stage II is $\rho_2 = 30.77\%$.

The system robustness for this scenario is quantified as $(\rho_1, \rho_2) = (74.5\%, 30.77\%)$.

V. Conclusions

The goal of this research is to study the allocation of resources to applications and the completion of their execution before a system deadline in the presence of uncertainty in input data and in system availability. A combined dual-stage framework has been proposed towards this goal. The framework enables the robustness of resource allocation used in a first stage to be enhanced via the use of dynamic loop scheduling techniques used in a second stage. The usefulness of the framework has been demonstrated via a small scale, illustrative example.

Extensions to this work may consider the impacts of employing previously developed static [4] and dynamic [19] stochastic resource allocation heuristics in stage I, and other dynamic loop scheduling techniques in stage II [14]. Designing novel robust and scalable resource allocation heuristics to be employed in stage I is also a noteworthy extension to the present work. A study of the factors to be considered in guiding the choice of heuristics used in either stage is another potential extension of interest to the current work. Exploring the possible correlation between the availabilities for different processor types on the overall robustness of the system is also of interest for our future work because it would help in better quantifying the system robustness.

In future work, a larger scale problem will be used to demonstrate the necessity of more advanced RA heuristics in stage I. This larger scale problem will incorporate more applications, i.e., in a larger batch or in multiple batches, on a larger computing system, i.e., one with more processors and processor types. Probabilistic studies will be performed on this larger problem to determine the benefit of the CDSF on a range of application and system parameters.

Acknowledgments: This work is in part supported by the German Research Foundation (DFG) in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing," by the National Science Foundation (NSF) under grant numbers CNS-0905399 and NSF IIP-1127978, and by the Colorado State University George T. Abell Endowment.

References

[1] N. R. Satish, "Compile Time Task and Resource Allocation of Concurrent Applications to Multiprocessor Systems," Ph.D. dissertation, University of California at Berkeley, Berkeley, CA, USA, 2009.

[2] C. Boneti, R. Gioiosa, F. J. Cazorla, and M. Valero, *Parallel and Distributed Computing, Alberto Ros (Ed.)*. InTech, 2010, ch. 7. Using Hardware Resource Allocation to Balance HPC Applications. [Online]. Available: http://www.intechopen.com/articles/show/title/using-hardware-resource-allocation-to-balance-hpc-applications

[3] 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, Jul. 2004.

[4] 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, August 2008.

[5] I. Banicescu, F. Ciorba, and R. L. Cariño, "Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems," *International Symposium on Parallel and Distributed Computing (ISPDC 2009)*, vol. 0, pp. 129–132, 2009.

[6] S. Srivastava, I. Banicescu, and F. Ciorba, "Investigating the robustness of adaptive dynamic loop scheduling on heterogeneous computing systems," in *Parallel and Distributed Scientific and Engineering Computing Workshop 2010,* in the Proceedings of the *International Parallel and Distributed Processing Symposium (IPDPSW-PDSEC 2010)*, Apr. 2010, pp. 1–8.

[7] A. Aziz and H. El-Rewini, "Grid resource allocation and task scheduling for resource intensive applications," in *International Conference on Parallel*

Processing Workshops (ICPP 2006 Workshops), 2006.

[8] M.-J. Huguet and P. Lopez, "Mixed task scheduling and resource allocation problems," in *In International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR 2000)*, Padeborn, Germany, 2000, pp. 71–79.

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

[10] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[11] O. 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, 1977.

[12] A. Hurson, J. Lim, K. Kavia, and B. Lee, "Parallelization of DOALL and DOACROSS loops: A survey," *Advances in Computers*, vol. 45, 1997.

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

[14] R. L. Cariño and I. Banicescu, "Dynamic load balancing with adaptive factoring methods in scientific applications," *The Journal of Supercomputing*, vol. 44, pp. 41–63, 2008.

[15] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A method for scheduling parallel loops," *Communications of the ACM*, vol. 35, no. 8, pp. 90–101, Aug. 1992.

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

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

[18] 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.

[19] J. Smith, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, "Stochastic-based robust dynamic resource allocation in a heterogeneous computing system," *International Conference on Parallel Processing (ICPP 2009)*, 2009.