2011 International Conference on Parallel Processing Workshops

# Energy-Constrained Dynamic Resource Allocation in a Heterogeneous Computing Environment

B. Dalton Young[1], Jonathan Apodaca[2], Luis Diego Briceño[1], Jay Smith[1,3], Sudeep Pasricha[1,2], Anthony A. Maciejewski[1], Howard Jay Siegel[1,2], Bhavesh Khemka[1], Shirish Bahirat[1], Adrian Ramirez[1], and Yong Zou[1]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373, USA
E-Mail: dalton.young@colostate.edu

[3]DigitalGlobe
Longmont, CO 80503, USA

***Abstract*—Energy-efficient resource allocation within clusters and data centers is important because of the growing cost of energy. We study the problem of energy-constrained dynamic allocation of tasks to a heterogeneous cluster computing environment. Our goal is to complete as many tasks by their individual deadlines and within the system energy constraint as possible given that task execution times are uncertain and the system is oversubscribed at times. We use Dynamic Voltage and Frequency Scaling (DVFS) to balance the energy consumption and execution time of each task. We design and evaluate (via simulation) a set of heuristics and filtering mechanisms for making allocations in our system. We show that the appropriate choice of filtering mechanisms improves performance more than the choice of heuristic (among the heuristics we tested).**

## I. Introduction and Problem Statement

Energy consumption of servers and data centers is a growing concern (e.g., [BoE02,Koo07]). Some studies predict that the annualized cost of powering and cooling servers may soon exceed the annualized cost of equipment acquisition [KoB09], which could force some servers to operate under a constraint on the amount of energy used to complete workloads.

In this research, we study the problem of dynamically allocating a collection of independent tasks to a heterogeneous computing cluster (heterogeneous both in terms of performance and power efficiency) while considering energy. We assume that the system is often oversubscribed, as is the case for the Oak Ridge National Labs Extreme Scale Systems Center system under development [BrK11]. The goal is to maximize the number of tasks completed by their individual deadlines under a constraint on the total amount of energy used by the system. Our problem formulation is more complex than earlier approaches because we consider the combination of a heterogeneous cluster, a time-varying arrival rate for tasks that causes the system to be oversubscribed at times, tasks with individual deadlines, stochastic task execution times, an energy constraint to process a fixed number of tasks, and using the concept of robustness (described in Section IV) in the objective functions of some heuristics.

We approach this problem by deriving resource allocation heuristics and filtering mechanisms that are capable of leveraging the cluster heterogeneities to maximize the number of tasks completed under a given energy constraint. We then compare these heuristics via simulation. Two of our heuristics are adapted from the literature to our environment, while the third is a novel heuristic that attempts to balance each task's energy consumption and probability of completing by its deadline. Additionally, our two filter mechanisms can be generically applied to any heuristic to add energy-awareness and/or robustness-awareness. Our workload (described in Section III-B) consists of a dynamically-arriving mix of different task types (e.g., compute-intensive, memory-intensive). Our system is oversubscribed at times, so we cannot utilize certain energy-conserving techniques (described in Section III-A). Thus, our heuristics and filters are limited to controlling energy consumption via task-to-machine mapping and processor Dynamic Voltage and Frequency Scaling (**DVFS**).

In this paper, we make the following contributions: (a) we develop a model of robustness for this environment and validate its use in allocation decisions, (b) we present an adaptation of two existing heuristics to utilize robustness and account for an energy constraint while making task-to-machine assignments, (c) we present a new heuristic for use in our environment, and (d) we

This research was supported by the National Science Foundation under grant number CNS-0905339, and by the Colorado State University George T. Abell Endowment. This research used the CSU ISTeC HPC System supported by NSF Grant CNS-0923386.

1530-2016/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPPW.2011.58

demonstrate the utility of our generalized filter mechanisms via simulations, which show at least a 13% improvement in each heuristic due to filtering.

The remainder of this paper is organized as follows. The next section discusses a sampling of the most closely-related work. In Section III, we define the model of the compute cluster, workload, and energy consumption of the system. Based on this system model, we formally introduce the concept of robustness and derive a robustness measure suitable to this environment in Section IV. Section V describes the heuristics used in this study. Section VI then discusses our simulation setup, while Section VII presents and analyzes our simulation results. We conclude with Section VIII, wherein we discuss extensions to this research and future directions.

## II. Related Work

The problem of mapping dynamically-arriving tasks under an energy constraint is addressed in [KiS08]. That work focused on conserving battery life in an ad-hoc grid environment while completing as many high-priority tasks as possible, followed by as many low-priority tasks as possible. The environment in [KiS08] also used a bursty environment with oversubscription. However, our study is fundamentally different because we work with probability distributions representing uncertain task execution times, whereas the work in [KiS08] used scalar task execution times. Also, our study focuses on a cluster environment with a single energy constraint, where the work in [KiS08] focused on an ad-hoc grid with energy constraints on a per-component basis.

The research in [XiL08] uses Dynamic Voltage and Frequency Scaling (**DVFS**) within a real-time system where the tasks have uncertain execution times. Unlike our study, there is no energy constraint and the system is not oversubscribed. The work in [XiL08] also emphasizes the benefits of inter-task DVFS to take advantage of slack time, but our system is oversubscribed at times. Similarly, the research in [YuV08] attempts to maximize a mathematical model of Quality of Service (**QoS**) under an energy constraint, but it does so using DVFS to take up slack time in an undersubscribed system, whereas our system is oversubscribed some of the time.

In [AyM01], the authors use a multi-part solution to save energy in a dynamic, real-time system with periodic tasks. Like [AyM01], our environment is dynamic, but it is also oversubscribed. Additionally, we have the goal of completing as many tasks by their deadlines as possible under an energy constraint, whereas the study in [AyM01] has the goal of minimizing energy under the constraint of completing all tasks by their hard deadlines. Also, the solution in [AyM01] utilizes a static component to develop its schedules, while our heuristics are limited to immediate-mode operation (tasks are mapped immediately upon arrival [MaA99]).

Similarly, in [KiB07] a set of independent tasks with individual deadlines are dynamically allocated to a cluster while attempting to conserve energy. Where [KiB07] attempts to optimize the energy consumed under the constraint of completing all tasks by their deadlines, our environment has an energy constraint and we optimize the number of tasks completed. The work in [KiB07] uses deterministic task execution times and constant arrival rates with an undersubscribed system, where our research focuses on stochastic task execution times and a bursty arrival rate with the system oversubscribed during task-arrival bursts.

This group has previously studied dynamic resource allocation in [SmA10,SmC09]. This research uses some heuristics from [SmC09] and part of the robustness definition from [SmA10]. However, neither of these previous works deal with energy-aware scheduling. We have also studied the static allocation of independent tasks with a common deadline to optimize the energy consumed in [ApY11].

## III. System Model

### A. Cluster Configuration

Our model of a cluster allows the performance and power efficiency of each node in the cluster to vary substantially. That is, the system is heterogeneous because it may consist of compute nodes that are quite different from one another. **Machine performance** is defined in terms of the time required to execute a given task, i.e., a higher performance machine will execute a task in less time than a lower performance machine. The machine performance of the nodes in this cluster is assumed to be **inconsistent** [AlS00], i.e., because machine $A$ is faster than machine $B$ on one task does not imply that machine $A$ is faster for all tasks.

Our model assumes that a cluster consists of $\boldsymbol{N}$ heterogeneous compute nodes, each with a different number of multicore processors, different number of cores in each multicore processor, different set of available processor frequencies, different power consumption profile, and different power supply efficiency. Each compute node $i$ consists of $\boldsymbol{n\,(i)}$ multicore processors, where $n\,(i)$ varies from one to four among nodes. Each multicore processor in compute node $i$ has $\boldsymbol{c\,(i)}$ cores, where $c\,(i)$ varies from one to four among nodes. Figure 1 shows the hierarchy of nodes, multicore processors, and cores within our cluster.

We assume that each core $k$ in multicore processor $j$ processes tasks independently of the other cores, and all cores and multicore processors within a given compute node are homogeneous. In this study, we limit the size of our cluster to eight compute nodes ($N = 8$) to limit our

simulation execution times, but our proposed techniques can be easily extended to larger clusters of nodes.

Our hierarchical cluster model is directly applicable to the ISTeC Cray XT6m system currently in use at Colorado State University [CSU11]. While the current system is homogeneous across compute nodes, there are plans to add GPUs to a limited subset of the compute nodes as well as to continually grow the system with new compute nodes based on the technology available at the time of purchase. Thus this system will indeed be heterogeneous and follow our hierarchical model.

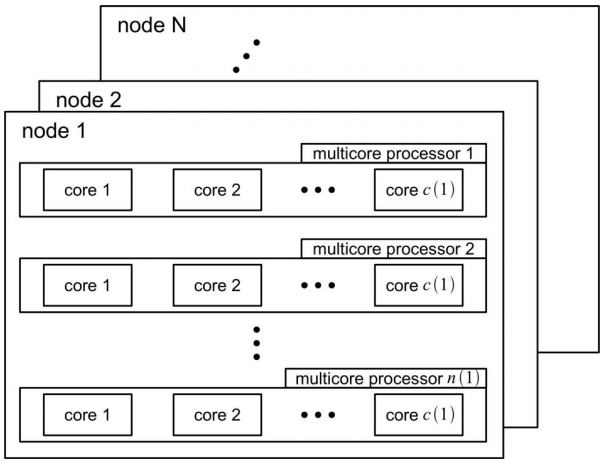


Fig. 1. The hierarchy of nodes, multicore processors, and cores that are used in our system model.

Our model of available processor frequencies and processor power consumption is based on the ACPI standard [Hew10]. The standard defines **P-states**, which are processor performance states that allow the processor to save power while executing instructions at the expense of decreased performance, i.e. increased task execution times. These states are used in DVFS implementations in many commodity processors (e.g., [Adv10b,Int10]).

Because our system is oversubscribed at unpredictable time intervals, we cannot turn nodes off. Thus, we assume that the energy consumed by the shared system components of each node (such as disk drives and fans) is constant within each node and can therefore be excluded from our computations (subtracted from the energy constraint before any tasks are scheduled). In this way, we are assuming that only the P-states can be used to save power.

The ACPI standard defines up to 16 P-states, but we will assume that a set of only five, denoted $\boldsymbol{P}$, is available: $P_0$, $P_1$, $P_2$, $P_3$, and $P_4$. Each P-state is associated with a certain clock speed and voltage configuration for a core that defines its corresponding power consumption. We will assume that, as power supplied to a core is increased, the performance of the core also will increase. Following convention [Hew10,Adv10c], let $P_0$ correspond to the base P-state that provides the highest power consumption (and therefore highest performance), and $P_4$ correspond to the P-state that provides the lowest power consumption (and therefore lowest performance). Power consumption in real systems can vary within a given P-state. For this study, we make the simplifying assumption that the power consumption of a core operating in a given P-state can be approximated by a scalar constant that represents the average power consumption. The power consumed by P-state $\pi$ in any core (because we assume that all cores and multicore processors within a compute node are identical) of compute node $i$ is denoted $\boldsymbol{\mu\,(i,\pi)}$. We will discuss the values of $\mu\,(i,\pi)$, as well as the relative performance of the cores in each P-state, in Section VI.

In our environment, the resource management system controls the P-states of each core individually. The operating system of each compute node provides a power management kernel that controls the P-state transitions for each core within each multicore processor, and we assume that cores within a multicore processor can switch P-states independently [Adv10c]. In this research, we assume that there is a cluster resource manager integrated with the operating system power manager so that the cluster resource manager can direct the power management kernel to change P-states, and that P-state transition times can be ignored because they are small (hundreds of microseconds [Adv10c]) with respect to task execution times (e.g., thousands of milliseconds). We also assume that cores can only change P-states when idle, i.e., P-states cannot be transitioned during task execution. The cluster resource manager will execute our resource allocation heuristics and take responsibility for controlling the power consumption of the cluster—in addition to assigning tasks to cores for execution.

Within a node of the cluster, power efficiency relates the total power provided by a power supply to the actual power it consumes. For example, a power supply with 90% efficiency supplies 90 watts of power to the elements of the node for every 100 watts of power it consumes. We assume that the efficiency of power supplies varies between nodes from at least 90% efficient to at most 98% efficient, and we denote the power efficiency of the power supply in node $i$ as $\boldsymbol{\epsilon\,(i)}$.

### *B. The Workload*

The workload in the environment in our model is a dynamically-arriving collection of independent tasks, i.e., the exact task mix is unknown prior to its arrival. However, each task is selected from a finite collection of well-known task types (such as may be found in many military, lab, government, and industrial environments). The different task types may stress different parts of the

system, i.e., some task types may be compute-intensive, others may be memory-intensive, etc.

In the workload, the execution time of each task is considered stochastic due to factors such as varying input data or cache conflicts, and that we are provided an execution-time probability mass function (**pmf**) for each task type executing on a single core of each node in each P-state (such pmfs may in practice be obtained by historical, experimental, or analytical techniques [LiA97,Was05]). More specifically, we model the execution time of each task type as a random variable and assume that we are given a probability mass function describing the possible execution time values and their probabilities for each task type, core, and P-state combination. We also assume that power consumption is a function of P-state and processor.

Our workload is characterized by a bursty arrival rate [LiB98], which will cause the system to be oversubscribed during burst periods and undersubscribed at other times. We assume that we are provided a deadline for completing each task $z$, denoted $\boldsymbol{\delta}(\boldsymbol{z})$, i.e., $\delta(z)$ defines a constraint on the completion time of task $z$. In a real system, task deadlines can come from multiple sources (e.g., limits set by system administrators, user requirements for timely data). Each deadline is considered a hard deadline, and there is no value in completing a task after its deadline has passed (i.e., the task is not counted as completed if its deadline is missed). This is simiilar to the system under development at the Extreme Scale Systems Center at Oak Ridge National Lab [BrK11]. We assume our cluster resource manager cannot stop a task after it has been scheduled and must execute it to completion as a best-effort basis, even if the task misses its deadline. We will describe our deadline and arrival rate models in more detail in Section VI.

The workload must consist of a finite number of tasks if an energy constraint is to be imposed. We test our heuristics over a "window" of 1,000 tasks. We assume that our resource management heuristics can make allocation decisions that take into account the number of tasks remaining in a "window." We also assume that our heuristics can query a processor for the number of tasks currently awaiting execution.

We limit our cluster resource manager to operating in an immediate-mode [MaA99]. Additionally, we assume that tasks cannot be reassigned, either to a new core or a new P-state, once they are mapped. Task mapping is controlled by the resource allocation heuristic employed by the cluster resource manager.

### C. Energy Consumption

As the tasks are independent and cores can change their P-states independently of one another, we can find the energy required by each core throughout the entire simulation independently of the other cores (recall that disk and memory energy requirements are treated as a constant and are therefore not included here). We can then find the total energy required by the cluster by summing the energy required by all cores. Because we assume that cores cannot be turned off, the energy consumption for each core can be found by identifying the time of each P-state transition, calculating the time difference between successive transitions, and multiplying this time difference by the power required to support that P-state.

Let $\boldsymbol{\nu(i,j,k)}$ denote the list of P-state transitions that are scheduled for core $k$ of multicore processor $j$ in node $i$ throughout the execution of the workload, let $\boldsymbol{|\nu(i,j,k)|}$ denote the size of list $\nu(i,j,k)$, let $\boldsymbol{\nu(i,j,k,n)}$ denote the $n^{th}$ P-state transition in list $\nu(i,j,k)$, let $\textbf{time}(\boldsymbol{x})$ denote the time of P-state transition $x$, and let $\textbf{pstate}(\boldsymbol{x})$ denote the ending P-state for the transition. We assume that each core makes at least two P-state transitions, one at the start of workload execution and one at the end of workload execution. If we define $\boldsymbol{\Delta t(n)} = time(\nu(i,j,k,n)) - time(\nu(i,j,k,n-1))$, we can compute the energy used by each core, which we denote $\boldsymbol{\eta(i,j,k)}$, as

$$\eta(i,j,k) = \sum_{n=1}^{|\nu(i,j,k)|} \mu(i, pstate(\nu(i,j,k,n))) \times \Delta t(n) \ . \tag{1}$$

Using the energy needed by each core ($\eta(i,j,k)$) from Equation 1, we can find the energy required to complete the entire workload, which we denote $\boldsymbol{\zeta}$, as

$$\zeta = \sum_{i=1}^{N} \sum_{j=1}^{n(i)} \sum_{k=1}^{c(i)} \frac{\eta(i,j,k)}{\epsilon(i)} \ . \tag{2}$$

We let $\boldsymbol{\zeta}_{\text{max}}$ denote the energy constraint for completing the workload.

## IV. Robustness

### A. Overview

We use random variables to model task execution times because the actual execution time of each task is uncertain [SmS09]. We want our resource allocations to be "robust," meaning that they mitigate the impact of uncertainties on our performance objective [AlM04]. More specifically, we want our resource allocations to mitigate the impact of the uncertain task execution times on our objective of completing as many tasks as possible, by their individual deadlines, within our energy constraint. This research builds on the robustness model presented in [SmA10].

When a system is described as robust, three questions must be answered [AlM08]: (1) What behavior makes

the system robust? (2) What uncertainties is the system robust against? (3) How is the robustness of the system quantified? In our system model, an allocation is robust if it can complete all tasks by their individual deadlines; an allocation is robust against uncertainties in task execution times; and the robustness of an allocation is quantified as the expected number of tasks which will complete by their deadlines.

### B. Stochastic Task Completion Time

At the $l^{th}$ time-step $t_l$, we want to predict the completion time of a task $z$ if it is assigned to a core $k$ in multicore processor $j$ of node $i$. Calculating this completion time requires combining the execution times for all tasks that have been assigned to core $k$ with the execution time of $z$. When using a deterministic (i.e., non-probabilistic) model, we calculate the completion time as the sum of the estimated execution times for all tasks assigned to core $k$, the estimated execution time for task $z$ if assigned to core $k$, and the ready time of core $k$. Because we are using a stochastic model (task execution times are represented by pmfs), we calculate the completion time as the sum of the random variables represented by the pmfs and the ready time. This sum requires a convolution of pmfs [Leo89,PhP03]. Convolutions can take considerable time, but the overhead can be negligible if task execution times are sufficiently long or the performance gained justifies their usage.

Let $\boldsymbol{Q(t_l)}$ be the set of all tasks that are either queued for execution or are currently executing on any of the cores in the cluster at time-step $t_l$. To determine the completion time of task $z$ if assigned to core $k$ of multicore processor $j$ on node $i$ at time-step $t_l$, we first identify the ordered list of tasks in $Q(t_l)$ that are assigned to core $k$, in order of their assignment to core $k$, and we let $\boldsymbol{Q(i,j,k,t_l)}$ denote this list. If there are no tasks assigned to core $k$, then $Q(i,j,k,t_l) = \emptyset$, and the ready time of this core is equal to the current time. In this case, the stochastic completion time of task $z$ if assigned to core $k$ is represented by the execution-time pmf of task $z$ on core $k$ in its chosen P-state, shifted by the current time.

If $Q(i,j,k,t_l) \neq \emptyset$, then the execution time pmf for the currently executing task $a$ on core $k$ (the first task in $Q(i,j,k,t_l)$) requires additional processing prior to its convolution with the pmfs of the queued tasks (other tasks in $Q(i,j,k,t_l)$). If $a$ began execution at time-step $t_h$ ($h < l$), some of the impulse values of the pmf describing the completion time of $a$ are in the past. Therefore, accurately describing the completion time of task $a$ at time $t_l$ requires shifting the execution-time distribution for task $a$ by $t_h$ (effectively creating the completion-time distribution for task $a$ if it started execution at time $t_h$), removing the past impulses from the pmf (those impulses which occur at time less than $t_l$), and re-normalizing the remaining distribution [SmC09]. After renormalization, the resulting distribution describes the completion time of $a$ on core $k$ as predicted at time-step $t_l$. This distribution is then convolved with the execution time pmfs of the tasks in $Q(i,j,k,t_l)$ and the execution-time pmf of task $z$ on this core in its chosen P-state to produce the completion-time pmf for task $z$ if assigned to core $k$ at the current time-step $t_l$.

The above computations can be applied to any task in $Q(i,j,k,t_l)$ to obtain a completion-time distribution. The completion-time distribution for the currently-executing task $a$ on core $k$, for example, can be found by shifting the execution-time distribution of $a$ assigned on core $k$ by its start time, removing impulses at time values less than $t_l$, and re-normalizing the distribution. The completion-time distribution for task $b$ assigned immediately after task $a$ on core $k$ can then be found by convolving the execution-time distribution of task $b$ assigned on core $k$ with the completion-time distribution of task $a$.

### C. Robustness Calculation

The robustness of a resource allocation in this environment is defined at a given time-step $t_l$ as the expected number of tasks that will complete by their individual deadlines, predicted at $t_l$ [SmA10]. We denote this value $\boldsymbol{\rho(t_l)}$. Because there is no inter-task communication in our cluster environment (all tasks are independent), the expected number of on-time task completions for all tasks assigned to a common core $k$ of multicore processor $j$ in node $i$ (all tasks in $Q(i,j,k,t_l)$) is independent across all cores and nodes. We let $\boldsymbol{\rho(i,j,k,t_l)}$ denote this value. If we let $\boldsymbol{\rho(i,j,k,\pi,t_l,q)}$ denote the probability of task $z$ in $Q(i,j,k,t_l)$ with P-state $\pi$ finishing by its deadline $\delta(z)$, then we can compute $\rho(i,j,k,t_l)$ as

$$\rho(i,j,k,t_l) = \sum_{\forall z \in Q(i,j,k,t_l)} \rho(i,j,k,\pi,t_l,z) \ . \quad (3)$$

We can therefore calculate $\rho(l)$ as

$$\rho(t_l) = \sum_{i=1}^{N} \sum_{j=1}^{n(i)} \sum_{k=1}^{c(i)} \rho(i,j,k,t_l) \ . \quad (4)$$

To complete as many tasks as possible by their individual deadlines, the research in [SmA10] indicates that we should maximize the expected number of on-time completions ($\rho(t_l)$) at each time-step [SmA10]. However, our system is limited to immediate-mode mapping. Therefore, if we are assigning a task $z$ at time-step $t_l$, we can maximize $\rho(t_l)$ by assigning $z$ to the core $k$ of multicore processor $j$ in node $i$ and P-state $\pi$ that maximizes $\rho(i,j,k,\pi,t_l,z)$ (assigning the task

where it has the highest probability of completing by its deadline).

To find a task's probability of completing by its deadline given a node, multicore processor, core, and P-state assignment, we can merely find the completion-time distribution for the assignments as described in Section IV-B, and then sum the impulses in the distribution that are less than the deadline. We will use this number in our filters and heuristics.

## V. Heuristics and Filters

### A. Overview

In this study, we adapted two task-scheduling heuristics taken from the literature to our cluster environment. We also created a new heuristic and implemented a random-assignment heuristic. An **assignment** consists of mapping a single task to a node, multicore processor, core, and P-state. Each heuristic, operating in an immediate-mode, assigns a single task to a node, multicore processor, core, and P-state for execution. We developed two filtering mechanisms that can be generically applied to any task-scheduling heuristic to limit the set of feasible assignments the heuristic may use. A filtering mechanism could eliminate all potential assignments, which would cause the task to remain unassigned and be discarded.

We use several expectation operations because our heuristics work with pmfs. For a task $z$ executing on core $k$ of multicore processor $j$ in node $i$ and P-state $\pi$, we compute the expected completion time (denoted ECT $\boldsymbol{(i,j,k,\pi,t_l,z)}$) by taking the expectation of the stochastic completion time distribution. Similarly, expected execution time (denoted EET $\boldsymbol{(i,j,k,\pi,z)}$) is found by taking the expectation of the execution-time distribution for a task. The expected energy consumption of an assignment (denoted EEC $\boldsymbol{(i,j,k,\pi,z)}$) is found by multiplying the expected execution time by the power consumption of the core and P-state where the task is assigned ($\mu(i,\pi)$), and then dividing by the power efficiency of the node where the task is assigned ($\epsilon(i)$).

### B. Shortest Queue Heuristic

The **Shortest Queue (SQ)** heuristic [SmC09] assigns the incoming task to the core with the fewest tasks currently assigned to it from among the feasible assignments. When invoked at time-step $t_l$, the heuristic finds the number of tasks currently assigned to each core $k$ of multicore processor $j$ in node $i$ in the list of feasible assignments (we denote this value $\boldsymbol{|MQ(i,j,k,t_l)|}$), and then maps the arriving task to the feasible assignment with the smallest value of $|MQ(i,j,k,t_l)|$. If multiple feasible assignments have the same minimum queue length, then the heuristic assigns the task to the core and P-state combination that has the minimum expected execution time for the task (EET $(i,j,k,\pi,z)$) among those with the minimum queue length.

### C. Minimum Expected Completion Time Heuristic

The **Minimum Expected Completion Time (MECT)** heuristic [MaA99], assigns the incoming task to the core and P-state combination that provides the minimum expected completion time from among the feasible assignments. When invoked at time-step $t_l$ to map task $z$, the heuristic finds the expectation of the completion-time distribution for task $z$ for each feasible assignment, i.e., ECT $(i,j,k,\pi,t_l,z)$. The heuristic then maps the task to the feasible assignment with the smallest expected completion time.

### D. Lightest Load Heuristic

The **Lightest Load (LL)** heuristic, our new herustic inspired by [BaM09], defines a load quantity, and then assigns the incoming task to the core and P-state combination that has the minimum load quantity from among the feasible assignments. We define load quantity as the product of the expected energy consumption and inverse robustness, and the heuristic then tries to minimize this product. When invoked at time-step $t_l$ to map task $z$, the heuristic first computes $\rho(i,j,k,\pi,t_l,z)$ and the expected energy consumption EEC $(i,j,k,\pi,z)$ for each feasible assignment. The LL heuristic then computes the load value for each potential assignment, denoted $\boldsymbol{L(i,j,k,\pi,t_l)}$, as

$$L(i,j,k,\pi,t_l) = \text{EEC}(i,j,k,\pi,z) \times (1.0 - \rho(i,j,k,\pi,t_l,z)) \ . \quad (5)$$

The heuristic then assigns the task to the feasible assignment with the smallest load value.

### E. Random Heuristic

The **Random** heuristic assigns the incoming task to a random core and P-state from among the feasible assignments. This is conceptually one of the simplest techniques for resource allocation, and we use it to contrast the benefits achieved by using the more sophisticated heuristics and our filter mechanisms.

### F. Energy and Robustness Filters

We use two filtering mechanisms to restrict the set of feasible assignments a heuristic can consider. These allow us to add energy-awareness and/or robustness-awareness to a heuristic that may have neither.

Our **energy filter** restricts the set of feasible assignments by eliminating all the potential assignments that would consume more than a "fair share" of the remaining energy budget as estimated at time-step $t_l$, which we denote $\boldsymbol{\zeta_{fair}(t_l)}$. A heuristic estimates the remaining energy as it runs by subtracting the expected

energy consumption of each assignment it makes from the energy budget for the simulation.

If we denote the heuristic's estimate of the remaining energy at time-step $t_l$ as $\boldsymbol{\zeta}(\boldsymbol{t_l})$ and the number of tasks that have not yet arrived as $\boldsymbol{T_{left}}(\boldsymbol{t_l})$, we can define a multiplier $\boldsymbol{\zeta_{mul}}$ and express our "fair share" threshold as

$$\zeta_{fair}(t_l) = (\zeta_{mul} \times \zeta(t_l)) / T_{left}(t_l) \ . \tag{6}$$

To deal with task-arrival bursts, we change $\zeta_{mul}$ based on the average queue depth of the system (the average of the number of tasks queued for execution or currently executing, at a single time-step). In our simulations, the best results were obtained using values of $\zeta_{mul} = 0.8$ for an average queue depth less than 0.8, $\zeta_{mul} = 1.0$ for an average queue depth of 0.8 to 1.0, and $\zeta_{mul} = 1.2$ for any average queue depth greater than 1.2.

Our **robustness filter** restricts the set of feasible assignments based on a probability threshold we denote $\boldsymbol{\rho_{thresh}}$. The filter eliminates potential assignments of task $z$ to core $k$ of multicore processor $j$ in node $i$ and P-state $\pi$ where $\rho(i,j,k,\pi,t_l,z) < \rho_{thresh}$ (i.e., potential assignments to node $i$, multicore processor $k$, core $j$, and P-state $\pi$ which will not increase the number of expected on-time completions by at least the probability threshold). Empirically, we determined that a threshold of $\rho_{thresh} = 0.5$ worked well for limiting the set of feasible assignments without restricting a heuristic to only high-performance (and therefore high energy consumption) P-state assignments.

## VI. Simulation Environment

A simulation trial in our environment consists of a collection of 1,000 tasks that arrive dynamically. Each task's type is selected uniformly at random from one of 100 task types. We generate a distribution describing the execution time of each task type on each machine using the CVB method described in [AlS00], with the parameters $\mu_{task} = 750$, $V_{task} = 0.25$, and $V_{mach} = 0.25$. Our entire simulation consists of 50 simulation trials with the task arrival times, task deadlines, and task types varying across simulation trials. All other parameters are held constant. Note that the simulated actual task execution times are randomly sampled from the execution time distributions during each trial, and so these vary across simulation trials.

In our simulations, we consider a bursty arrival rate [LiB98]. We use an early burst of task arrivals (first 200 tasks) and a late burst of task arrivals (last 200 tasks), with a lull (600 tasks) between bursts. This effectively makes the system undersubscribed between task arrival bursts, which allows heuristics and filters to try to conserve energy. Our task arrivals follow a Poisson process, and we define an **equilibrium rate $\boldsymbol{\lambda_{eq}}$** such that the system is perfectly subscribed (all tasks complete by their deadlines with no energy to spare) if all tasks arrive following a Poisson process with this rate (in our simulations, $\lambda_{eq} = 1/28 \approx 0.0357$). From this parameter, we then define a **fast rate $\boldsymbol{\lambda_{fast}}$** and a **slow rate $\boldsymbol{\lambda_{slow}}$** such that task arrivals following a Poisson process with the fast rate will cause the system to be oversubscribed, and task arrivals following a Poisson process with the slow rate will cause the system to be undersubscribed (in our simulations, $\lambda_{fast} = 1/8 = 0.125$, and $\lambda_{slow} = 1/48 \approx 0.0208$). With these parameters, we generate all the task arrivals for a single simulation trial using a Poisson process where the rate is $\lambda_{fast}$ for the first 200 tasks, $\lambda_{slow}$ for the next 600 tasks, and $\lambda_{fast}$ for the last 200 tasks. This means that the arrival rates are constant across simulation trials, but the arrival times may vary considerably.

For our simulation study, task deadlines are set for each task as the sum of its arrival time, the average execution time of its task type over all machines and all P-states, and a constant "load factor." The "load factor" represents the anticipated waiting time of a task before it begins execution. We assign deadlines assuming that each task will not have to wait longer than the **average task execution time $\boldsymbol{t_{avg}}$**, which we compute as the average execution time of all task types across all machines and all P-states (in our simulations, $t_{avg} \approx 1353$). We can then define the load factor as $t_{avg}$; the actual load will be higher when the arrival rate is $\lambda_{fast}$, and lower when the arrival rate is $\lambda_{slow}$.

The clock-speed profile for the five P-states of each core is specified by a set of five multipliers that scale the execution time distributions of a task executing on that core to reflect a higher or lower performance. The multipliers for each P-state are calculated by adding a random sample from a uniform distribution to the previous multiplier (in effect increasing the performance by a random percentage between 15% and 25%). For all cores, the minimum P-state multiplier was never less than 42% of the maximum, implying that the minimum operating frequency was at least 42% of the maximum (there are current AMD Phenom processors with similar frequency ratios [Adv10a]).

The power consumption profile for the five P-states of each core is calculated using the standard CMOS dynamic power dissipation formula (recall that power used by other system components is assumed constant and has already been accounted for in the energy constraint). If $A$ is the number of transistor switches per clock cycle, $C_L$ is the capacitive load, $V$ is the supply voltage, and $f$ is the operating frequency, then the capacitive power dissipation is

$$P_c = A \times C_L \times V^2 \times f \ . \tag{7}$$

We first calculate the power consumption in the highest

P-state for each core by sampling a uniform random distribution between 125 and 135 Watts. We then sample a uniform random distribution between 1.000 and 1.150 to get a low P-state voltage, sample a uniform random distribution between 1.400 and 1.550 to get a high P-state voltage, calculate the voltage numbers for the remaining P-states via linear interpolation, factor $A \times C_L$ into a constant, and use our voltage and frequency values for each P-state to compute a power consumption value. In practice, this results in a power consumption for the low P-state of about 25% that in the high P-state (again, there are current AMD Phenom processors with similar power consumption values [Adv10a]).

If we let $\boldsymbol{p_{avg}}$ denote the average power over all P-states and all machines, then

$$p_{avg} = \frac{1}{N \times |P|} \sum_{i=1}^{N} \sum_{\forall \pi \in P} \mu\left(i, \pi\right) \ . \tag{8}$$

The energy constraint for our simulation ($\zeta^{\max}$) is $t_{avg} \times p_{avg} \times 1,000$, which would be the energy required to execute an average task one thousand times. Because the deadlines are tight, this amount of energy will be insufficient to finish all tasks by their deadlines, which will force heuristics to make a trade-off.

## VII. Results

Box-and-whiskers plots for the results of each heuristic and its variations are shown in Figures 5 through 4, while Figure 6 shows the simulation results for the best variation of each heuristic. In each figure, "none," "en," "rob," and "en+rob" represent the results of the heuristic with no filtering, energy filtering only, robustness filtering only, and both energy and robustness filtering, respectively.

We first note from Figures 2 and 3 that the unfiltered versions of the SQ and MECT heuristics ("none") have median values of 375.5 missed deadlines (37.55% of 1,000) and 370 missed deadlines (37% of 1,000), respectively. These are both slightly better than the unfiltered LL heuristic (Figure 4), which has a median value of 381 missed deadlines (38.1% of 1,000). The unfiltered Random heuristic (Figure 5) is noticeably worse, with a median value of 561.5 missed deadlines (56.15% of 1,000). The MECT and SQ heuristics perform poorly without filtering because they make no attempt to conserve energy: MECT will choose $P_0$ to get a smaller completion time, and SQ will choose $P_0$ to get a minimum execution time when breaking ties. LL performs poorly because, during congestion, all the robustness values fall and LL will choose minimum-energy assignments to minimize the load until the congestion clears.

The results for the Random heuristic, as seen in Figure 5, bear some explanation because they differ from the trend set by all other results. With the LL, SQ, and

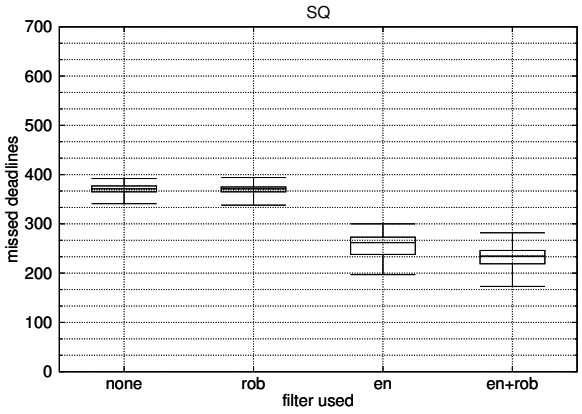


Fig. 2. The number of missed deadlines for all variations of the SQ heuristic are shown with a box-and-whiskers plot.

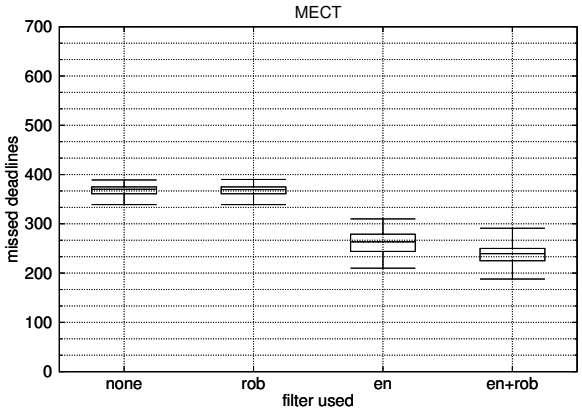


Fig. 3. The number of missed deadlines for all variations of the MECT heuristic are shown with a box-and-whiskers plot.

MECT heuristics, energy filtering helps considerably while robustness filtering is only useful when combined with energy filtering. With the Random heuristic, however, energy filtering ("en") alone actually worsens the median performance by 3.45%. This is because the assignments are selected uniformly at random from among the feasible assignments, and using energy filtering removes the high-performance assignments from the potential list. Notice that the robustness filter ("rob") provides a large benefit for the Random heuristic (a 22.6% improvement in median performance from 561.5 to 335.5 misses) for an analagous reason: it removes the low-performance assignments from the potential assignment list.

Figures 2 through 5 show that using robustness filtering without energy filtering ("rob") causes no significant change in results for heuristics other than Random, but using robustness filtering with energy filtering ("en+rob") consistently improves the median performance by two to three percent over energy filtering alone

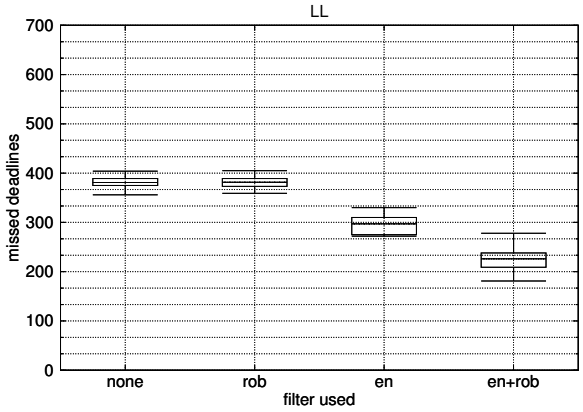


Fig. 4. The number of missed deadlines for all variations of the LL heuristic are shown with a box-and-whiskers plot.

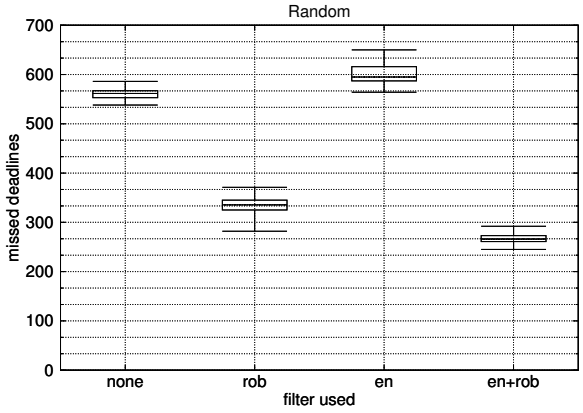


Fig. 5. The number of missed deadlines for all variations of the Random heuristic are shown with a box-and-whiskers plot.

("en"). It is not surprising that robustness alone is a poor filtering mechanism. Consider robustness filtering used with the MECT heuristic: the filter eliminates all potential assignments with less than a 50% chance of completing the task by its deadline, but this has very little impact on the normal operation of MECT. This is because MECT automatically chooses the highest P-state as that guarantees the fastest execution time for the task (and therefore faster completion time than the same assignment with any other P-state).

If we then compare the best variation of all the heuristics, we can see from Figure 6 that the energy-filtered and robustness-filtered LL heuristic ("en+rob") has the best median performance (226 missed deadlines, or 22.6% of 1,000). The filtered MECT heuristic ("en+rob") misses a median of 239.5 task deadlines (23.95% of 1,000), and the filtered SQ ("en+rob") heuristic misses 234.5 task deadlines (23.45% of 1,000). Thus, all heuristics exhibit an improvement of at least 13% over their unfiltered counterparts. Our LL heuristic exhibits a 14%

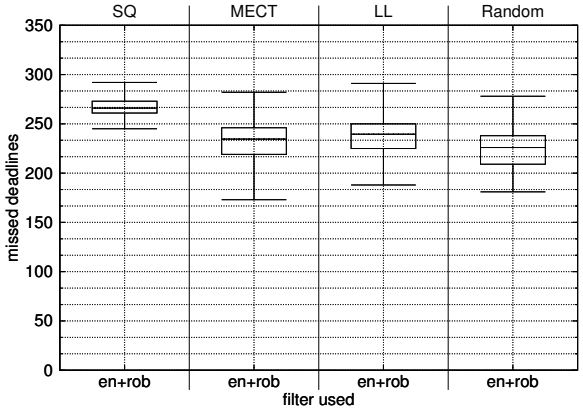


Fig. 6. The number of missed deadlines for the best-performing variation of each heuristic are shown with a box-and-whiskers plot.

improvement over an unfiltered MECT heuristic. The "en+rob" Random heuristic actually improves its median misses by nearly 30% over the unfiltered version (266 missed deadlines compared to 561.5 missed deadlines), and is only 4% from the "en+rob" LL heuristic. This demonstrates that filters actually drive the performance.

In summary, consider the importance of the filtering operations. Using "en+rob" filtering for the Random, SQ, MECT, and LL heuristics results in improvements in the number of tasks completed by their individual deadlines within the energy constraint of 25%, 13.65%, 13.05%, and 15.5%, respectively. Additionally, filtering allows the Random heuristic to come within 4% of the complex LL heuristic. These results demonstrate how filtering is the key to success in our environment.

## VIII. Conclusions and Future Work

In this paper, we developed a model of robustness for our environment and validated its use in allocation decisions. We also presented two filters, two adaptations of existing heuristics, and one new heuristic that can make assignments utilizing robustness and accounting for an energy constraint. Based on our simulation results, we can conclude that appropriate filtering mechanisms are fundamental to improving heuristic performance in our environment.

In future work, we would like to extend our model to include more energy-conserving techniques than just DVFS. This would include mechanisms such as ACPI G-states, power gating, and hard-disk power management. We also want to use full probability distributions to represent power consumption, instead of assuming that power consumption is a constant representing an average value (as described in Section VI). We intend to expand our model to consider tasks with varying priorities, and a system with the ability to cancel and/or reschedule tasks. Finally, we want to include a variety of arrival

rates and patterns, to better understand how the relative performance of the heuristics changes under varying conditions.

**Acknowledgments:** The authors thank Greg Pfister and Jerry Potter for their comments on this research.

## REFERENCES


[Adv10a] Advanced Micro Devices, *AMD Family 10h Desktop Processor Power and Thermal Data Sheet*, Rev 3.46, 2010, http://support.amd.com/us/Processor_TechDocs/43375.pdf, accessed March 2, 2011.

[Adv10b] Advanced Micro Devices, *AMD PowerNow! Technology*, 2010, http://www.amd.com/us/products/technologies/amd-powernow-technology/Pages/amd-powernow-technology.aspx, accessed March 2, 2011.

[Adv10c] Advanced Micro Devices, *BIOS and Kernel Developer's Guide (BKDG) for Family 10h Processors*, Rev 3.48, 2010, http://support.amd.com/us/Processor_TechDocs/31116.pdf, accessed March 2, 2011.

[AlM04] 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, July 2004, pp. 630–641.

[AlM08] S. Ali, A. A. Maciejewski, and H. J. Siegel, *Perspectives on Robust Resource Allocation for Heterogeneous Parallel Systems*, Chapman & Hall/CRC Press, Boca Raton, FL, 2008, pp. 41–1–41–30.

[AlS00] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering, Special $50^{th}$ Anniversary Issue*, Vol. 3, No. 3, Nov. 2000, pp. 195–207.

[ApY11] J. Apodaca, D. Young, L. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou, "Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment," *$9^{th}$ ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '11)*, Dec. 2011.

[AyM01] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," *$22^{nd}$ IEEE Real-Time Systems Symposium (RTSS '01)*, Dec. 2001, pp. 95–105.

[BaM09] J. Barbosa and B. Moreira, "Dynamic job scheduling on heterogeneous clusters," *$8^{th}$ International Symposium on Parallel and Distributed Computing (ISPDC '09)*, July 2009, pp. 3–10.

[BoE02] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," *Power Aware Computing*, Kluwer Academic Publishers, Norwell, MA, USA, 2002, pp. 261–289.

[BrK11] L. D. Briceño, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing system," *$4^{th}$ Heterogeneity in Computing Workshop (HCW '11)*, May 2011, pp. 1–14.

[CSU11] CSU Information Science and Technology Center, *ISTeC Cray High Performance Computing (HPC) System*, 2011, http://istec.colostate.edu/istec_cray, accessed June 15, 2011.

[Hew10] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation Std., *Advanced Configuration and Power Interface Specification*, Rev. 4.0a, Apr. 2010, http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf, accessed March 2, 2011.

[Int10] Intel Corporation, *Frequently asked questions for Intel SpeedStep Technology*, 2010, http://www.intel.com/support/processors/sb/CS-028855.htm, accessed March 2, 2011.

[KiB07] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters," *$7^{th}$ IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '05)*, May 2007, pp. 541–548.

[KiS08] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 11, Nov. 2008, pp. 1445–1457.

[KoB09] J. G. Koomey, C. Belady, M. Patterson, A. Santos, and K.-D. Lange, "Assessing trends over time in performance, costs, and energy use for servers," Lawrence Berkeley National Laboratory, Stanford University, Microsoft Corporation, and Intel Corporation, Tech. Rep., Aug. 2009, accessed March 2, 2011, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5562&rep=rep1&type=pdf.

[Koo07] J. G. Koomey, "Estimating total power consumption by servers in the U.S. and the world," Lawrence Berkeley National Laboratory, Tech. Rep., Feb. 2007, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5562&rep=rep1&type=pdf.

[Leo89] A. Leon-Garcia, *Probability & Random Processes for Electrical Engineering*, Addison Wesley, Reading, MA, 1989.

[LiA97] 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, July 1997, pp. 35–52.

[LiB98] C. Li, R. Bettati, and W. Zhao, "Response time analysis for distributed real-time systems with bursty job arrivals," *1998 International Conference on Parallel Processing (ICPP '98)*, Aug. 1998, pp. 432–440.

[MaA99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–121.

[PhP03] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems, and Transforms*, Pearson Education, Upper Saddle River, NJ, 2003.

[SmA10] J. Smith, J. Apodaca, A. A. Maciejewski, and H. J. Siegel, "Batch mode stochastic-based robust dynamic resource allocation in a heterogeneous computing system," *2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '10)*, July 2010, pp. 263–269.

[SmC09] J. Smith, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, "Stochastic-based robust dynamic resource allocation in a heterogeneous computing system," *$38^{th}$ International Conference on Parallel Processing (ICPP '09)*, Sep. 2009.

[SmS09] J. Smith, H. J. Siegel, and A. A. Maciejewski, "Robust resource allocation in heterogeneous parallel and distributed computing systems," *Wiley Encyclopedia of Computer Science and Engineering*, B. W. Wah, Ed., John Wiley & Sons, Hoboken, NJ, 2009, Vol. 4, pp. 2461–2470.

[Was05] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*, Springer Science+Business Media, New York, NY, 2005.

[XiL08] C. Xian, Y.-H. Lu, and Z. Li, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, No. 8, Aug. 2008, pp. 1467–1478.

[YuV08] H. Yu, B. Veeravalli, and Y. Ha, "Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems," *2008 Asia and South Pacific Design Automation Conference (ASPDAC '08)*, Mar. 2008, pp. 452–455.