Heterogeneous Energy and Makespan-Constrained DAG Scheduling

Bobby Dalton Young
Colorado State University
Dept. of Electrical & Computer Engineering
Fort Collins, Colorado 80523
dalton.young@colostate.edu

Sudeep Pasricha
Colorado State University
Dept. of Electrical & Computer Engineering
Fort Collins, Colorado 80523
sudeep@colostate.edu

Anthony A. Maciejewski
Colorado State University
Dept. of Electrical & Computer Engineering
Fort Collins, Colorado 80523
aam@colostate.edu

Howard Jay Siegel
Colorado State University
Dept. of Electrical & Computer Engineering
Dept. of Computer Science
Fort Collins, Colorado 80523
HJ@colostate.edu

James T. Smith
Lagrange Systems
1426 Pearl St, #207
Boulder, Colorado 80302
jay@lagrangesystems.com

ABSTRACT

Energy-efficient resource allocation within computing systems is important because of the growing demand for, and cost of, energy. In this paper, we study the problem of energy-constrained static resource allocation of a collection of communicating tasks to a heterogeneous computing environment. Our goal is to maximize the probability (calculated via Monte Carlo method) that our collection of tasks completes by both a given deadline and an energy constraint in an environment where task execution times and communication times are uncertain. We model a collection of energy-saving mechanisms from the ACPI standard that can be used to balance the energy consumption and execution time of our tasks. We then design and evaluate (via simulation) a set of heuristics for allocating resources in our system. Finally, we show that our novel adaptation of existing heuristics can greatly improve performance in our environment.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*scheduling*

Keywords

Static Resource Allocation; Energy-Aware Scheduling

EEHPDC'13, June 17, 2013, New York, NY, USA.

1. INTRODUCTION

Energy consumption of servers and data centers is a growing concern [1,2]. Some studies predict that the annualized cost of powering and cooling servers may soon exceed the annualized cost of equipment acquisition [3]. This could force some computing systems to operate under a constraint on the amount of energy used to complete workloads, especially given the rising cost of energy and the energy requirements of upcoming exascale systems [4,5].

In this research, we study the problem of statically allocating a collection of communicating tasks to a heterogeneous computing system (heterogeneous across compute nodes in terms of both performance and power efficiency) while considering energy. The communicating tasks form a Directed Acyclic Graph (**DAG**), and our goal is to maximize the probability of completing all the tasks in this DAG under two constraints: a system deadline and a system energy consumption budget. The problem of allocating a set of dependent tasks with a common deadline to a heterogeneous system while accounting for energy consumption has been explored before, e.g., [6–8]. We study the same problem, but we include stochastic task execution and communication times, a broader set of energy-conserving techniques derived from the ACPI standard [9], a simple mechanism to approximately model the effects of slowdown caused by interference between cores of the multicore processors, and using the concept of robustness (described in Section 4) in the objective functions of some heuristics.

We approach this problem by first modeling a heterogeneous computing system, and then deriving resource allocation heuristics that are capable of leveraging the system heterogeneities to maximize the probability of completing all tasks by both a common deadline and a given energy constraint. We then compare the performance of these heuristics via simulation. Our three heuristics are inspired by concepts in the existing literature [10]: two are adaptations to our environment needed to explore the solution space while the third is a novel variation designed to directly balance

completion time and energy consumption. Our DAG model (described in Section 3.2) consists of a graph of dependent tasks from the literature [11]. Our heterogeneous computing platform model provides a range of energy-conserving mechanisms for heuristics to use (described in Section 3.3). These include 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 provide a method to compute the energy characteristics of our system that we believe captures the salient attributes of energy consumption without being overly complex, (b) we design a model of robustness specifically for this environment and demonstrate its usefulness for scheduling, (c) we design and present three heuristics for our environment (inspired by concepts in the previous literature), one of which utilizes our robustness model and attains significantly better performance than the others, and (d) we design a model of our heterogeneous computing environment that simulates the effects of slowdown due to interference between tasks executing on different cores.

2. RELATED WORK

There has been considerable research regarding scheduling DAGs onto heterogeneous systems with an emphasis on robustness (a good survey can be found in [12]). Our research, however, focuses specifically on energy-aware allocations, and our definition of robustness is more relevant to our domain.

Similar to the work in [6], we use DVFS to balance power consumption and computation time of tasks in a DAG executing on a heterogeneous system. However, we combine this with stochastic task execution and communication delay times. We also consider the energy consumption of other system components (e.g., memory, NIC) with energy-conserving operating modes. In this way, we can capture many of the salient attributes of a real computing system with a model that is still widely applicable.

The authors of [8] study the allocation of a DAG with a deadline (in the form of timing constraints) in a real-time system. They develop "shared slack reclamation" heuristics to conserve energy while scheduling the DAG to meet timing constraints. The authors then prove some formal properties of the algorithms and simulate these heuristics over a collection of important DAGs. In our study, we work with stochastic execution times instead of known (deterministic) scalar values. We also include energy-conserving mechanisms in addition to DVFS, and we approximate the effects of interference between the cores of multicore processors.

The research in [7] is also related to our work. The authors focus on energy and makespan-constrained DAG allocation on a heterogeneous system, but without the stochastic nature of our study. They are also able to build an effective algorithm (similar to the work in [13]), but our system model includes more of the energy-saving mechanisms present on current systems.

3. SYSTEM MODEL

3.1 System Configuration

We model our computing system as a collection of heterogeneous compute nodes with **inconsistent** performance [14] (i.e., machine A may be faster than machine B for some tasks and slower for others). Each compute node i contains a set of homogeneous multicore processors, and each multicore processor j contains a set of homogeneous cores, so performance is heterogeneous across nodes and homogeneous within each node. For convenient notation, let $\boldsymbol{n}$ denote the total number of compute nodes, $\boldsymbol{m(i)}$ denote a list of all the multicore processors in node i, $\boldsymbol{c(i)}$ denote a list of all the cores in node i, and $\boldsymbol{c(i,j)}$ represent a list of all the cores in multicore processor j of compute node i.

Cores are the basic processing elements in our model. Our model allows each core to execute a single task at a time with no multithreading or preemption, as is the case on the compute nodes of the ISTeC Cray XT6m teraflop computing system at Colorado State University [15]. In this way, a task will execute to completion once started.

Even though the cores of each multicore processor execute single tasks without preemption or multithreading, they still interfere with each other by sharing processor features, e.g., shared L3 cache. We model the effects of this interference by increasing the execution time of temporally-overlapping tasks executing on the same multicore processors. More specifically, the execution time of a task is increased by a fixed percentage while simultaneously executing with a task on another core of the same multicore processor. This effect is cumulative in our model, so each additional task executing on another core of the same multicore processor will increase the execution time of all other tasks due to memory interference. We use a fixed-percentage increase based on our specific workload and system environment, but other values could be used. This approach follows real-world observations that the actual slowdown due to multicore interference is highly workload-dependent and system-specific [16,17].

Cores can vary the execution speed of tasks by using Dynamic Voltage and Frequency Scaling (DVFS). Based on the ACPI standard [9] implemented in many commodity processors [18,19], our model provides each core with a set of performance states (**P-states**) that decrease instruction execution speed at the expense of increased core energy consumption. We allow the individual cores of a multicore processor to change P-states independently (as allowed in many current processors such as those from AMD [20]), but we disallow P-state transitions once a task has started executing to minimize the energy overhead of frequent P-state transitions. We also ignore the time needed to transition between P-states because that time is small (typically hundreds of microseconds [20]) compared to our task execution times (typically tens of seconds) and communication times (typically seconds).

In addition to the multicore processors, each compute node in our model contains a Network Interface Card (**NIC**) used to facilitate communication between cores on different compute nodes. We model the communication between nodes as a sum of two components: a delay time representing network congestion and packet collisions between the two nodes and a transfer time needed to move each unit of data from one compute node to another multiplied by the total amount of data to be sent. To model an uncertain communication time, we modify the existing work to use a stochastic delay time with a deterministic unit-transfer time. More specifically, if we denote the deterministic time required to move one unit of data from any core on compute node i to any core on a different compute node m as $t_{unit}(i,m)$ (a

scalar value) and the stochastic delay caused by congestion and collisions in the network between the same two compute nodes as $t_{delay}(i,m)$ (a probability mass function or **pmf**), then the time required to transfer u units of data between compute nodes i and m is:

$$t_{comm}(i,m) = (n \times t_{unit}(i,m)) + t_{delay}(i,m) \ . \quad (1)$$

We assume that cores can send and receive data while computing, but our model currently requires synchronous communication between compute nodes. We currently restrict each compute node to send to only one compute node at a time and receive from only one compute node at a time, but the send and receive operations can happen simultaneously even to different compute nodes.

In addition to the multicore processors and NICs, our system model also includes a disk and memory in each node. These are included to model energy consumption of the system more accurately and are described in Section 3.3.

All of the compute nodes are managed by a single resource manager that constructs a static (offline) schedule according to a resource allocation heuristic and then executes tasks on the computing system according to the schedule. A schedule consists of a mapping of each task to a compute node, multicore processor, core, and P-state. Each schedule also includes an ordering of the communications between compute nodes. In our current model, the ordering for both tasks and communication is strictly followed at run time.

3.2 DAG of Tasks

In our environment, the DAG consists of a static collection of tasks with a single common deadline and an energy constraint (budget) for execution. Dependencies among the tasks (e.g., communications) are represented by the edges of a DAG. We model task execution times as random variables, and our model assumes that a pmf that describes the execution time of each task in each P-state on a core of each compute node is provided (in practice, such pmfs can be generated from historical data, experimental trials, or analytical techniques [21,22]). It is common in resource management research to assume that information characterizing the execution times of frequently-executed tasks can be collected, e.g., [23,24]. In our collaborative work with Oak Ridge National Labs, their environment (as well as others) frequently executes similar types of tasks. This allows for the collection of historical information about task execution times.

The dependencies between tasks in the DAG represent communication in our model, meaning that each task consumes data from its predecessors and provides data to its successors. The amount of data sent from each task to its successors is fixed and known in advance. We make the common assumptions that the data provided by a predecessor task is only available once the task has completed and that all data to be consumed by a successor task must be received before computation can begin, e.g., [25].

We model a hard deadline for completion of all tasks in the DAG (such deadlines can come from multiple sources, e.g., QoS agreements, allocation limits). Similarly, we model an energy constraint to complete the entire workload. We focus on a single DAG in this paper, but it is straightforward to apply the techniques proposed in this paper to multiple DAGs that share a common deadline.

3.3 Energy Consumption

The energy consumed by the entire computing system can be conceptualized as the sum of the energy consumed by the system components over all of the individual time periods when those components are not changing state. Because of the dependencies mentioned in Section 3.2, some compute nodes may be idle while others are still processing tasks, and we account for this by measuring energy consumption on all compute nodes in our system from the time that any task in the DAG begins execution until the time that all tasks in the DAG have finished executing. This approach provides more realistic results than only measuring energy consumption when compute nodes are active, but may result in idle components consuming a large portion of the total system energy if the DAG does not have enough parallelism to keep the compute nodes busy. In this section, we describe both the mechanisms our model provides to mitigate the energy consumption of idle components and our method for computing the total energy consumption of the system.

As mentioned in Section 3.1, the cores inside our compute nodes can vary the execution speed of tasks by using P-states from the ACPI standard [9]. We assume that every core in the system has $|\rho + 1|$ P-states available. In accordance with the standard, we let P_0 denote the state with the highest performance and highest power consumption, and P_ρ denote the state with the lowest performance and lowest power consumption.

In addition to P-states, our model provides two power states (**C-states**) for each multicore processor to conserve power if and only if all of its cores are idle [9]. We let C_0 denote the normal operating C-state in which the cores are executing instructions, and C_1 denote the low-power C-state in which all cores are idle. While operating in C_0, any idle cores can minimize processor power consumption by operating in P_ρ. If any core in a multicore processor is not idle, then the multicore processor cannot enter the C_1 state to conserve power. This behavior is consistent with many current-generation AMD processors [20]. As with the P-states mentioned in Section 3.1, we neglect the time required to transition between C-states because it is negligible compared to average task execution times for our workloads. Based on our observations of real processors, we model the power consumption of a multicore processor idling in C_1 as a fixed smaller percentage of the power consumption of the same multicore processor with all cores active and operating in P_ρ. The model can be adapted to other systems by changing this parameter.

Our model also includes energy consumption of the memory, hard disk, and NIC devices of each node. Our model provides two device states (**D-states**) for each of these devices to conserve power when idle: D_0 denotes the active state, and D_1 denotes the idle state [9]. As in most systems, our model considers the memory to be idle if all cores in the compute node associated with the memory are idle (i.e., the multicore processors are all in C_1) and the NIC to be idle if it is neither sending nor receiving data. We model the hard disk as active half of the time when memory is active to represent virtual memory access.

At the compute-node level, our model provides two global states (**G-states**) to allow idle nodes to conserve power [9]. We let G_0 denote the state in which a compute node has started processing tasks on some core but has not finished all its tasks and G_1 represent the low-power state in which a

compute node has either not started processing tasks or has completed processing all its tasks. We model the compute-node energy consumption in G_1 as a small percentage of the energy consumption in G_0 with all devices in D_1 and all multicore processors in C_1 (i.e., their idle states).

We allow compute nodes to enter G_1 only before processing any tasks or after finishing all tasks. This may seem to be a somewhat artificial restriction: the compute node could certainly be suspended while completely idle between processing tasks. However, the time and energy overhead required to suspend and resume a compute node is non-negligible and does not save energy (in fact in some cases it increases the energy consumed).

The power consumption of a processor normally varies even within a given P-state. For all of the following calculations, we assume that power consumption of a core in a specific P-state represents the time-average power consumption of the system components for that P-state.

Consider a single core k of multicore processor j in node i from time t_a to t_b operating in P-state P_{ijk} (where P_{ijk} must be P_ρ if the core is idle). If we let $p(P_{ijk})$ denote the power consumed by a core of compute node i operating in P-state P_{ijk}, then we can denote the energy consumed by this core as:

$$en(i,j,k,t_a,t_b) = (t_b - t_a) \times p(P_{ijk}) \ . \tag{2}$$

If we let $pct(i,j,C_x)$ denote the fixed-percentage multiplier used to compute the energy consumed in C_x as described above ($pct(i,j,C_0) = 1$ and $0 < pct(i,j,C_1) < 1$), then we can use this along with equation 2 to compute the energy consumed by a multicore processor j in node i operating in state C_{ij} from time t_a to t_b as:

$$\begin{aligned} &en(i,j,t_a,t_b) \\ &= pct(i,j,C_{ij}) \times \sum_{\forall k \in c(i,j)} en(i,j,k,t_a,t_b) \ . \end{aligned} \tag{3}$$

The energy consumption equations for the devices in the system (disks, memory, NICs) are very similar, so only the equation for a NIC is shown here. If we let $p_{NIC}(i,D_x)$ denote the power consumed by a NIC in node i operating in D-state D_x, then the energy consumed by a NIC in node i operating in D-state D_i from time t_a to t_b is:

$$en_{NIC}(i,t_a,t_b) = (t_b - t_a) \times p_{NIC}(i,D_i) \ . \tag{4}$$

Let $en_{dev}(i,t_a,t_b)$ represent the sum of the energy consumed by all the devices (NIC, memory, disk) in node i from time t_a to time t_b. If $pct(i,G_x)$ denotes the fixed-percentage multiplier used to compute the energy consumed in G_1 as mentioned above (such that $pct(i,G_0) = 1$ and $0 < pct(i,G_1) < 1$), then the energy consumption of node i operating in G-state G_i from time t_a to t_b is:

$$\begin{aligned} &en(i,t_a,t_b) = pct(i,G_i) \\ &\times \left(en_{dev}(i,t_a,t_b) + \sum_{\forall j \in m(i)} en(i,j,t_a,t_b) \right) \ . \end{aligned} \tag{5}$$

Finally, we can write the system energy consumption from time t_a to t_b as:

$$en(t_a,t_b) = \sum_{x=0}^{n} en(x,t_a,t_b) \ . \tag{6}$$

The above equations are impacted by the uncertainty in communication and computation times described in Section 3.1. We will discuss this effect in more detail in Section 4.

4. ROBUSTNESS

Because task execution and communication delay times may be uncertain [26–28], we use random variables to model these times. We want our resource allocations to be "robust," meaning that they mitigate the impact of these uncertainties on our performance objective [29,30]. We can more-precisely describe the robustness of our system by answering three questions [31]: (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 in the DAG by both the common deadline and the energy consumption constraint. The system is robust against uncertainties in task execution time and communication delay time. The robustness of the system is quantified as the probability of completing all tasks by the deadline and within the energy constraint.

We need to maximize the probability that all tasks will be completed by the two constraints. This probability can be directly computed from a completion-time distribution pmf, but directly computing the completion-time distribution of a scheduled DAG consisting of nodes with stochastic execution times is known to be #P-complete (computationally intractable) [13,32]. To circumvent this problem, we use a Monte Carlo method.

To compute the robustness of a schedule, we need to compute the makespan and energy consumption. We first sample the pmfs for task-execution time and communication-delay time to obtain scalar values. We then use those values to compute the start and stop times for each task and each node-to-node communication in the system, all while scaling the schedule for interference (as described in Section 3.1). Once this is finished, we can compute the completion time of the schedule as well as the energy consumption (using the equations in Section 3.3).

By repeating the above procedure 1000 times to find completion time and energy consumption pairs, we are essentially performing a Monte Carlo evaluation of the schedule. Each repetition randomly samples the distributions and provides another (completion time, energy consumption) pair for the schedule. By comparing each pair to the deadline and energy constraint, we can quantify our robustness as the percentage of Monte Carlo trials that met both the energy constraint and deadline. The variation in communication and execution times results in considerable variation in the energy consumption across Monte Carlo evaluations.

Figure 1 shows a graphical representation of the robustness calculation. Using the procedure desribed above, we find the completion time and energy consumption one thousand times for a given assignment (schedule) of the DAG to the system. Each pair of values is plotted as a single point, and the percentage of pairs falling within both the energy constraint (horizontal line) and deadline (vertical line) is the robustness of the assignment.

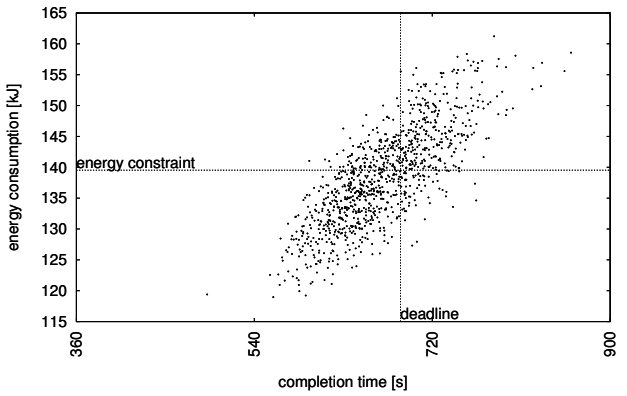

Figure 1: An example robustness computation showing the results of 1000 Monte Carlo evaluations with a given schedule. The result of each Monte Carlo evaluation is shown as a single point. The percentage of points falling within both the energy constraint (the horizontal dotted line) and the deadline (the vertical dotted line) is equal to the robustness of the schedule.

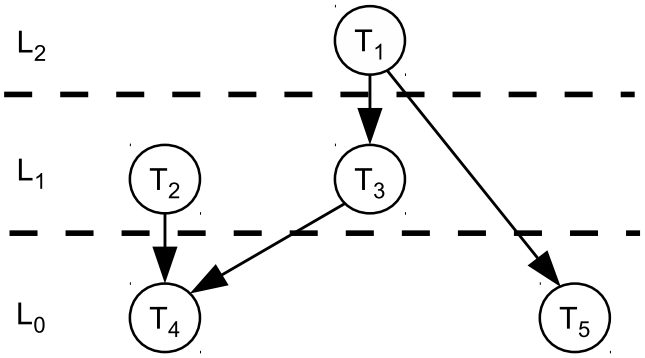

Figure 2: An example of levelizing a DAG consisting of five tasks. Each task is represented by a circle, and data sent from a task to a successor is represented by a directed edge connecting two tasks. Each task T_1 through T_5 is placed in one of the three levels L_0 through L_2 (demarcated by the dashed lines) based on its highest-level successor task.

5. HEURISTICS

5.1 Overview

In this study, we designed three heuristics based on the concepts of the "Bottoms Up" (BU) task-scheduling heuristic proposed in the literature [33]. The first two heuristics are necessary to explore the search space and are used for comparison to the third. All variations share the two-phase greedy list-scheduling approach of the BU heuristic. Basically, each heuristic assigns one task of the DAG at a time until the entire DAG is scheduled. Each assignment simply consists of mapping the task to a node, multicore processor, core, and P-state for execution. After each iteration, we can evaluate the makespan, energy consumption, and partial robustness of the partially-scheduled DAG (the set of scheduled tasks) as described in Sections 3.3 and 4 to make scheduling decisions. The concept of a two-phase heuristic has been successfully used in other environments, e.g., [14,34].

The BU heuristic (as described in [10]) requires a **levelized** DAG. The levelizing procedure assigns each task of the DAG to a level based on its precedence constraints. We first assign each task with no successors to level 0 (the highest-priority level). We then iterate through the tasks, assigning each to a level one higher than the highest-level consumer of its output data. In Figure 2, tasks 4 and 5 have no successors, and are assigned to level 0. Tasks 2 and 3 produce data consumed on level 0, so they are assigned to level 1. Task 1 produces data consumed on levels 0 and 1, so it is assigned to level 2.

5.2 Bottoms Up Two-Phase Heuristics

The three heuristics have different objectives, but use a similar procedure for scheduling. Each heuristic schedules each task using two phases to greedily minimize (or maximize) the value of its objective for the partially-scheduled DAG. In the first phase, each heuristic iterates through all the tasks in the current level. For each task, a copy of the current partial schedule is made for every combination of node, multicore processor, and P-state with the task scheduled on that node, multicore processor, core, and P-state. All copies are then evaluated using the Monte Carlo approach, and the combination that resulted in the minimum (or maximum) objective value for the partial schedule is stored for that task. In the second phase, the heuristic chooses the combination with the minimum (or maximum) objective value from among the stored combinations and schedules the task with that combination accordingly. The objective of the **Bottoms Up Min-Min Completion Time (CT)** heuristic is to minimize completion time, while the objective of the **Bottoms Up Min-Min Energy (En)** heuristic is minimize energy consumption. The objective of the **Bottoms Up Max-Max Robustness (Rb)** heuristic is to maximize robustness directly.

The Rb heuristic only adds one task to the schedule during each iteration, so the robustness calculations described in Section 4 cannot be applied directly. This is because the first several tasks assigned will meet both constraints regardless of the quality of their assignments, and therefore the robustness of all potential assignments for these tasks will be 100%. We address this problem by using the completion-time and energy-consumption values from the Monte Carlo evaluation to compute the percentage of the deadline and percentage of the energy constraint remaining after each potential assignment. We then sum these two values and pick the potential assignment with the greatest sum. In theory, this approach would allow a schedule 0% remaining energy (energy constraint exceeded) and 80% remaining deadline (completion time at 20% of the deadline) to be chosen over a schedule with 30% remaining of both the energy constraint and the makespan, but in practice this situation never arises. Because the heuristics only map one task per iteration, all the schedules compared during a single iteration of the heuristic differ by only one task assignment and therefore have similar values for the percentage of deadline and percentage of energy budget remaining.

There is also a limitation to the En heuristic. Consider the DAG in Figure 2 again, and suppose that only task 4 has been scheduled. The multicore processor executing task

4 must have all of its cores active in at least the lowest-performance P-state (as described in Section 3.3). Because of this overhead from other cores, executing task 5 on another core of the same multicore processor as task 4 will result in a lower energy consumption than executing it on a different multicore processor in the same compute node. Similarly, executing task 5 on another compute node will incur the energy overhead for the idle cores in its multicore processor as well as the energy overhead for other components in the node (as described in Section 3.3). These overheads bias the assignment of task 5 towards the same multicore processor as task 4, which may result in a higher energy consumption in the final schedule than if task 5 was assigned to its true minimum-energy machine. The Rb heuristic shares this behavior, but also has the advantage of optimizing for both completion time and energy consumption and is therefore less-affected.

6. SIMULATION ENVIRONMENT

All of our simulation trials use the structure of the 88-task robot-control DAG [35] from the Standard Task Graph set [11]. All trials also share a common system configuration of 28 cores divided into four compute nodes: a node with four dual-core processors, a node with three dual-core processors, a node with two tri-core processors, and a node with two quad-core processors. Additionally, the simulation trials share a single matrix of communication delay-time pmfs (one entry for each pair of compute nodes) and a single set of data transfer requirements for the tasks in the DAG. Our entire simulation consists of a set of 48 simulation trials, with only the matrix of execution-time pmfs (one entry for each task on each compute node) varying across trials.

To make our system heterogeneous, task execution-time pmfs are generated using the CVB method described in [36] with the parameters V_{task} and V_{mach} set to 0.25 and μ_{task} equal to the scalar value provided as the execution time of each task in the Standard Task Graph robot-control DAG file [11]. In this way, we keep the average task execution time across all machines roughly equal to the values from the original data source, but different compute nodes will have different execution-time pmfs for the same task.

The robot-control DAG does not include communication-time information between tasks, so we generate it. First, we make every task transmit data packages (between one and eight, selected uniformly at random) to each of its children. We then take the average number of packages transferred and pick a value for the deterministic transfer time (see Section 3.1) such that the deterministic transfer time needed to send all data packages in the DAG is roughly half of the total expected time needed to compute all tasks in the DAG. We then generate a set of random numbers from a normal distribution with a mean equal to the deterministic transfer time and a variance of 0.25, and these numbers are used for the deterministic transfer times for each pair of compute nodes (the values of $t_{unit}(i, m)$ from Equation 1).

With the data transfer amounts and deterministic transfer times, we generate the matrix of communication delay-time pmfs (the values of $t_{delay}(i, m)$ from Equation 1) using the CVB method [36] with the parameters $V_{task} = V_{mach} = 0.25$ and μ_{task} equal to the deterministic transfer time. In this way, the total communication time is roughly equal to the total computation time. In reality, though, there will be less communication than computation because tasks scheduled on the same compute node do not need to communicate.

For our simulation study, the deadline and energy constraint for the workload are set using the En and CT heuristics (these would be determined by the computing facility in practice). We set the deadline equal to the minimum average completion time of the En heuristic over 48 simulation trials and the energy constraint equal to the minimum average energy consumption of the CT heuristic over 48 simulation trials. By doing this, we effectively limit the best-case robustness of these two heuristics to be at most 50%. These constraints are loose enough, however, to give our Rb heuristic room to trade energy consumption for completion time.

In our simulations, we assume all multicore processors have five P-states for each core. 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 [37]). We approximated the effects of multicore interference (as mentioned in Section 3.1) using a fixed-percentage multiplier of 5%.

The power consumption profile for the five P-states of each core is extrapolated using real data with the standard CMOS dynamic power dissipation formula. This implicitly assumes that dynamic power is the major component of processor power consumption, but we are using the formula for extrapolation based on real data, so this assumption has little effect. 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 dynamic power dissipation is

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

We first calculate the power consumption in the highest P-state for each core by randomly sampling a uniform distribution between 125 and 135 Watts (values taken from datasheets [37]). We then randomly sample a uniform distribution between 1.000 and 1.150 to obtain a low P-state voltage, randomly sample a uniform distribution between 1.400 and 1.550 to obtain 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% of that in the high P-state (again, there are current AMD Phenom processors with similar power consumption values [37]).

Power consumption for system components (NIC, disk, and memory) is computed by sampling a uniform distribution around a datasheet value for each D-state. The D_0 and D_1 mean values for disks are 9.5W and 7W, respectively. The D_0 and D_1 mean values for memory are 4W and 2W, respectively. The D_0 and D_1 mean values for the NICs are 1.5W and 6W, respectively.

As mentioned in Section 3.3, the power consumed in C-

state C_1 is a fixed percentage of the power consumed with all cores idle in P_4. We study the results when this multiplier is 5%, 25%, and 50%. There are current AMD Phenom processors with a C_1 power consumption around 25% of the lowest C_0 value [37], and we try 5% and 50% to examine the extent to which C_1 energy consumption affects the total energy consumption of the system. Similarly, we use a G_1 multiplier of 10% to isolate the effects of idle compute nodes. The power consumption profile for the NIC, disk, and memory in both device states are sampled from uniform random distributions based on current datasheet values.

We use 1000 iterations in each of our Monte Carlo evaluations. This value works well empirically, as the average completion time of all trials is within 1% of the average at 10,000 evaluations.

7. RESULTS

Box-and-whiskers plots for the results of each of the three heuristics for each of the three C_1 multipliers (described in Section 6) are shown in Figure 3(a). Each plot shows the minimum, first quartile, median, third quartile, and maximum robustness values obtained for the heuristic over the 48 trials.

As described in Section 6, the mean energy consumption of the lowest-energy-consumption CT trial is used to set the energy constraint, and the mean completion time of the shortest completion-time En trial is used to set the deadline. Because of this, we immediately see that the best-case robustness of the CT and En heuristics barely exceeds 50% in the best case (the trial used to set the constraints). The Rb heuristic very clearly considers both constraints, resulting in a median robustness of 88% in the most-realistic case (C_1 multiplier is 25%). Even in the worst case (C_1 multiplier of 50%), the Rb heuristic still obtains a median robustness of $\approx$ 75% (nearly 60% higher than that of the En heuristic).

It is also interesting to note from Figure 3(a) that the En heuristic performs very poorly when the C_1 multiplier is 5%. As the multiplier increases, the En heuristic schedules tasks to execute more quickly because leaving compute nodes active (or even idle) uses considerably more energy. At the 5% multiplier, though, the En heuristic schedules nearly every task to run very slowly, which results in a slightly-lower ($\approx$ 9%) median energy consumption than the other heuristics, but a much longer ($\approx$ 30%) median execution time. This long execution time makes it nearly impossible for the En heuristic to meet the completion-time constraint, and so the robustness of the heuristic is nearly 0%.

To better understand these results, we first examine the completion time (shown in Figure 3(b)) and energy consumption (shown in Figure 3(c)) of the same 48 trials.

In Figure 3(b), the makespan of the En heuristic decreases as the C_1 multiplier increases. This indicates clearly that, as the energy consumption for idle components (the C_1 energy consumption) increases, minimizing energy consumption requires lowering the makespan. This happens because energy consumption is measured until all tasks in the DAG have completed, and so a schedule with a longer makespan consumes more energy in idle components. As the C_1 multiplier increases, that extra energy becomes a significant part of the total energy consumed by the system. We can also observe that, while the energy consumption of all heuristics increases as the C_1 multiplier increases, the energy consumption of the CT heuristic increases much more dramatically

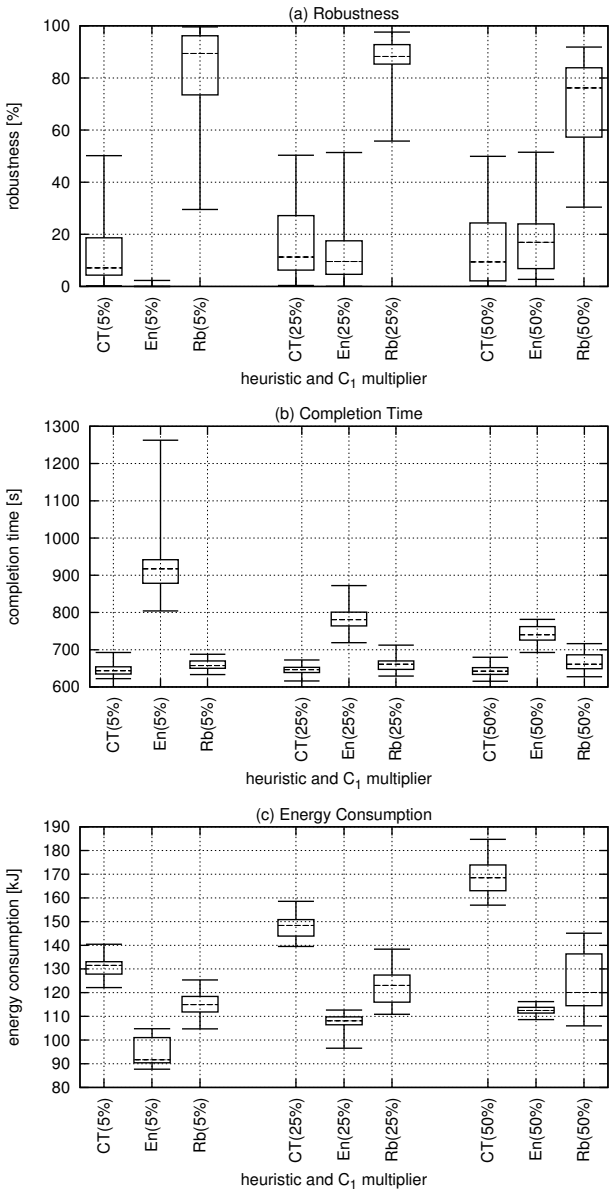

Figure 3: Box-and-whiskers plots showing the minimum, first quartile, median, third quartile, and maximum values obtained for each of the three heuristics at each C_1 multiplier over the 48 simulation trials. Recall that the C_1 multiplier represents the percentage of idle C_0 energy consumed by the multicore processors while in C_1. In (a), the Rb heuristic always obtains a median robustness at least 60% higher than any other heuristic. In (b), we see that the completion time of the En heuristic decreases drastically as the C_1 multiplier increases, indicating that the energy consumption during C_1 is significant. In (c), the Rb heuristic always obtains an energy consumption between that of the CT and En heuristics, indicating its successful tradeoff behavior.

than that of the other two. This indicates that the schedules produced by the CT heuristic include large amounts of idle time in the system.

If we examine the schedules produced by the three heuristics for a single representative trial, we can see each heuristic optimizing directly towards its objective. The CT heuristic will spread tasks among compute nodes freely to minimize completion time, achieving a completion time of about 650s. Likewise, the En heuristic will assign all tasks to the cores of the most energy-efficient compute node, leaving all other nodes in G_1 (the low-power state for an entire compute node), resulting in a lower energy consumption with a completion time of about 800s. The Rb heuristic makes an intelligent tradeoff by using only the same energy-efficient compute node as the En heuristic for most of the tasks, but placing a small group of tasks on a fast compute node to decrease completion time. This results in a completion time of just over 700s, but with a 25% energy savings over the CT heuristic.

Finally, we can examine the energy consumption and completion time of all trials (Figure 4). This plot shows the standard deviation in energy consumption and completion time as an ellipse around the mean of each trial over 1000 Monte Carlo evaluations. Each of the 48 trials is connected, and we clearly see the Rb heuristic trading between the CT and En heuristics to stay within the constraints.

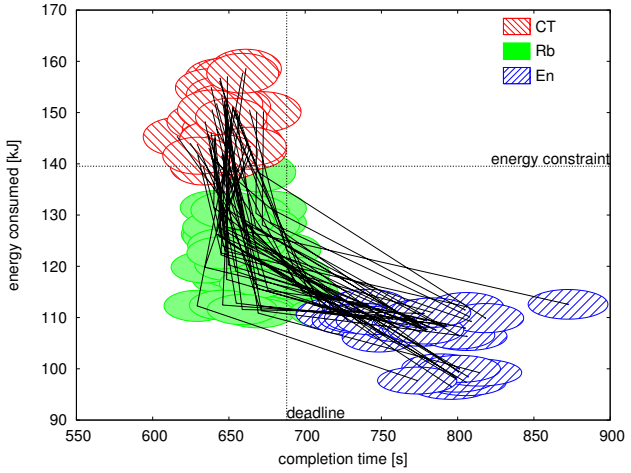

Figure 4: The means and standard deviations, indicated by the center and major/minor axes of the ellipses respectively, of each of the three heuristics over all 48 trials at a C_1 multiplier of 25%.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we designed an energy consumption model for our system. We also designed a definition of robustness for our environment and validated its use in allocation decisions via simulation. We presented two adaptations of existing heuristics, and one novel heuristic that can make assignments utilizing robustness and accounting for an energy constraint. Based on our results, we can conclude that our new heuristic greatly improves performance in our environment.

For future work, we first want to consider other DAGs and system configurations. In particular, the Standard Task Graph set [11] contains application DAGs for a sparse matrix solver and the fpppp routine from the SPEC benchmark. These graphs differ considerably from the robot DAG we use, and testing our model and heuristics with them will be useful. We can also vary the amount of communication in each DAG and study the behavior of our heuristics. We would also like to vary our system configuration to examine how a larger or smaller number of nodes, multicore processors, and cores affect the scheduling of the different DAGs.

We also plan to implement some new heuristics and improve our existing ones for further testing. We have plans to implement a Genetic Algorithm, which will allow us to compare our greedy heuristics to an iterative heuristic.

9. ACKNOWLEDGMENTS

This research was supported by the NSF under grant number CNS-0905399, and by the Colorado State University George T. Abell Endowment. This research used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386. The authors thank Greg Pfister and Jerry Potter for their comments on this research.

10. REFERENCES

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

[2] I. Rodero, A. Quiroz, M. Parashar, F. Guim, and S. Poole, "Energy-efficient Online Provisioning for HPC Workloads", in *Handbook of Energy-Aware and Green Computing*. Boca Raton, FL: Chapman & Hall/CRC Press, 2012.

[3] 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. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5562&rep=rep1&type=pdf

[4] N. Leavitt, "Big iron moves toward exascale computing," *Computer*, vol. 45, pp. 14–17, 2012.

[5] M. Tolentino and K. Cameron, "The optimist, the pessimist, and the global race to exascale in 20 megawatts," *Computer*, vol. 45, no. 1, pp. 95–97, Jan. 2012.

[6] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *9th IEEE/ACM Int'l Symp. on Cluster Comput. and the Grid (CCGrid '09)*, 2009, pp. 92–99.

[7] H. Sheikh and I. Ahmad, "Simultaneous optimization of performance, energy and temperature for dag scheduling in multi-core processors," in *3rd Int'l Green Comput. Conf. (IGCC '12)*, June 2012, pp. 1–6.

[8] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 7, pp. 686–700, July 2003.

[9] *Advanced Configuration and Power Interface Specification*, Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and

Toshiba Corporation Std., Rev. 4.0a, Apr. 2010. http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

[10] S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad-hoc grid environment," *J. Parallel Distrib. Comput.*, vol. 66, no. 4, pp. 600–611, Apr. 2006.

[11] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Scheduling*, vol. 5, no. 5, pp. 379–394, Sep. 2002.

[12] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics," in *Grid Comput.* Springer, 2008, pp. 73–84.

[13] L.-C. Canon and E. Jeannot, "Evaluation and optimization of the robustness of dag schedules in heterogeneous environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 532–546, Apr. 2010.

[14] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

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

[16] X. Xiang, B. Bao, C. Ding, and K. Shen, "Cache conscious task regrouping on multicore processors," in *12th IEEE/ACM Int'l Symp. on Cluster Comput. and the Grid (CCGRID '12)*, May 2012, pp. 603–611.

[17] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Design, Autom. Test in Europe Conf. Exhibition (DATE '10)*, Mar. 2010, pp. 741–746.

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

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

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

[21] 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," *J. Parallel Distrib. Comput.*, vol. 44, no. 1, pp. 35–52, Jul. 1997.

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

[23] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *Computer*, vol. 26, no. 6, pp. 18–27, Jun. 1993.

[24] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and contention-aware multi-resource reservation," *Cluster Comput.*, vol. 4, no. 2, pp. 95–107, Apr. 2001.

[25] J.-J. Han and D.-Q. Wang, "Edge scheduling algorithms in parallel and distributed systems," in *Int'l Conf. Parallel Process. (ICPP '06)*, Aug. 2006, pp. 147–154.

[26] A. Dogan and F. Özgüner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Comput.*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

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

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

[29] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 7, pp. 630–641, Jul. 2004.

[30] L. Bölöni and D. Marinescu, "Robust scheduling of metaprograms," *J. Scheduling*, vol. 5, no. 5, pp. 395–412, Sep. 2002.

[31] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems," in *Handbook of Parallel Computing: Models, Algorithms, and Applications.* Boca Raton, FL: Chapman & Hall/CRC Press, 2008, pp. 41-1–41-30.

[32] J. N. Hagstrom, "Computational complexity of PERT problems," *Networks*, vol. 18, no. 2, pp. 139–147, June 1988.

[33] N. Mehdiratta and K. Ghose, "A bottom-up approach to task scheduling on distributed memory multiprocessors," in *23rd Int'l Conf. Parallel Process. (ICPP '94)*, vol. 2, Aug. 1994, pp. 151–154.

[34] 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," *J. Parallel Distrib. Comput.*, vol. 59, no. 2, pp. 107–121, Nov. 1999.

[35] H. Kasahara and S. Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," *IEEE J. Robot. Autom.*, vol. 1, no. 2, pp. 104–113, June 1985.

[36] 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 50th Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

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