Energy and Deadline Constrained Robust Stochastic Static Resource Allocation

Mark A. Oxley[1], Sudeep Pasricha[12],
Howard Jay Siegel[12], and Anthony A. Maciejewski[1]

[1] Department of Electrical and Computer Engineering
[2] Department of Computer Science
Colorado State University, Fort Collins, Colorado 80523, USA
{mark.oxley,sudeep,hj,aam}@colostate.edu

Abstract. In this paper, we study the problem of energy and deadline constrained static resource allocation where a collection of independent tasks ("bag-of-tasks") is assigned to a heterogeneous computing system. Computing systems often operate in environments where task execution times vary (e.g., due to data dependent execution times), therefore we model the execution time of tasks stochastically. This research focuses on the design of energy-constrained resource allocation heuristics that maximize robustness against the uncertainties in task execution times. We design and evaluate a new resource allocation heuristic based on Tabu Search that employs dynamic voltage and frequency scaling (DVFS) and exploits heterogeneity by incorporating novel local search techniques.

Keywords: heterogeneous computing; static resource allocation; power-aware computing; DVFS; robustness

1 Introduction

The electricity used by data centers has increased by 56% worldwide between the years 2005 and 2010 [10]. This increase highlights the need for energy-aware resource management for these data centers. Many of these data centers form heterogeneous computing environments that utilize a mixture of different machines to execute workloads with diverse computational requirements. In this work, we assume that nodes are heterogeneous in both performance and power consumption, where the performance of individual nodes in the cluster are inconsistent in the following sense: if machine A is faster than machine B for a given task, machine A may not be faster for all tasks.

Resource allocation decisions often rely on estimated values for task execution times where actual values may differ from available estimates (e.g., due to data dependent execution times). To account for these variations, we model task execution times stochastically using random variables.

This research addresses the problem of *statically* allocating a workload of independent (non-communicating) tasks to a high-performance heterogeneous computing cluster. We aim to develop and analyze resource management techniques that maximize *robustness*, the probability that our workload finishes by a common deadline, while maintaining a specified probability that the energy consumed is within a given energy budget.

We make three main contributions in this work. Our first contribution is the design of three new intelligent local search techniques that maximize robustness within an energy constraint for a heterogeneous computing environment. The second contribution is an analysis of the impact of each local search technique's ability to improve a solution and the benefits associated with combining the local search techniques. The third contribution is the design and analysis of two global search techniques and the effects of combining global and local search into a new Tabu Search heuristic.

2 Related Work

Robust resource allocation has been previously studied (e.g., [2, 3, 13, 14]). We adopt the system model from [2], however that work addresses the problem of minimizing energy consumption with a robustness constraint, where in our work we attempt to maximize robustness under an energy constraint and change the robustness calculation to be more generally applicable. The other studies listed do not consider energy when optimizing for robustness. Our work uses energy-aware resource allocation techniques and DVFS to balance the tradeoff between performance and energy consumption to maximize robustness under an energy constraint.

The use of DVFS to balance performance and energy consumption has been studied widely (e.g., [5, 11, 15]). In [5], the focus is on minimizing energy consumption, rather than maximizing performance, and considers deterministic rather than stochastic execution times as we do in our work. DVFS is used to tradeoff between energy consumption and makespan in [11], where the workload consists of precedence-constrained applications represented by a directed acyclic graph (DAG). Scheduling is performed on a task-by-task basis using a novel objective function that considers both energy consumption and the estimated finishing time of a task, using DVFS to exploit slack times from precedence constraints. Our work considers stochastic execution times, robustness, the probability of meeting an energy budget, and the overhead energy consumption of compute nodes. Using DVFS to schedule tasks with uncertain execution times is addressed in [15], however the system model is homogeneous and the work does not consider an energy constraint.

3 System Model

3.1 Compute Nodes

Our compute cluster model, adopted from [2], consists of N heterogeneous compute nodes where the performance and power consumption of each node may vary substantially. Nodes are also heterogeneous in their processor configuration.

Each compute node i consists of $\boldsymbol{n_i}$ multicore processors, and each multicore processor j in compute node i has $\boldsymbol{n_{ij}}$ cores. We assume that all cores and multicore processors in a given compute node are homogeneous. We use the triple *ijk* to denote core k on multicore processor j in compute node i.

We assume the processors are DVFS-enabled to use the Advanced Configuration and Power Interface (**ACPI**) performance states (**P-states**) [7] that allow the processor to change operating voltage and frequency. We assume each core has a set of five P-states, denoted $\boldsymbol{P}$, available: P_0, P_1, P_2, P_3, and P_4. Each core in the system can operate in an individual P-state, and our resource allocation techniques are designed such that P-states do not switch during task execution. Lower-numbered P-states consume more power, but provide better performance. The power consumption of a core within compute node i in P-state P_π is $\boldsymbol{\rho_i^{(\pi)}}$.

3.2 Workload

The workload in this environment consists of a collection of $\boldsymbol{T}$ independent tasks to be completed before a given system deadline $\boldsymbol{\delta}$. Such a collection of independent tasks is known as a *bag-of-tasks* [6], the primary workload in various types of distributed computing systems [9]. We assume the task execution times follow a Gaussian distribution, and the means and variances are known *a priori* so that we can perform a static (i.e., off-line) mapping. The use of Gaussian distributions allows us to sum task execution times using a closed-form equation rather than having to perform convolution, however our proposed resource management techniques are applicable for task execution times that follow any distribution with a calculable mean (expected value). In an actual system, the means and variances of the execution time distributions can be obtained by historical, experimental, or analytical techniques [13]. We work closely with Oak Ridge National Labs, and in their environment, as well as others, similar types of tasks are executed frequently allowing for the collection of historical information of the execution times of tasks on machines.

For a given resource allocation, let the set $\boldsymbol{T_{ijk}}$ denote tasks in T that have been assigned to core *ijk* and let $\boldsymbol{t_{ijk}^x} \in \boldsymbol{T_{ijk}}$. Let $\mathbf{PS}(\boldsymbol{t_{ijk}^x})$ denote the assigned P-state for task $\boldsymbol{t_{ijk}^x}$. We denote the mean execution time associated with task $\boldsymbol{t_{ijk}^x}$ in P-state $\boldsymbol{\pi}$ as $\boldsymbol{\mu(t_{ijk}^x, \pi)}$, and the associated variance as $\boldsymbol{V(t_{ijk}^x, \pi)}$.

3.3 Energy Constraint Calculation

The energy required to execute tasks in a single core can be calculated as the product of the power consumption of the core in each P-state and the amount of time the core spends in each P-state. Let $\boldsymbol{T_{ijk}^{\pi}}$ denote the subset of tasks assigned to core *ijk* in P-state $\boldsymbol{\pi}$. The energy consumed is calculated as the product of execution time (a random variable) and average power (a deterministic value). Multiplying a random variable with a known value has the effect of multiplying the expected value by that value and the variance by the square of that value. For core *ijk*, the expected value ($\boldsymbol{S_{ijk}^{\pi}}$) and variance ($\boldsymbol{Var_{ijk}^{\pi}}$) are

$$S^{\pi}_{ijk} = \sum_{t^x_{ijk} \in T^{\pi}_{ijk}} \rho_i^{(\pi)} \mu(t^x_{ijk}, \pi) \quad (1) \text{ and } \quad Var^{\pi}_{ijk} = \sum_{t^x_{ijk} \in T^{\pi}_{ijk}} (\rho_i^{(\pi)})^2 V(t^x_{ijk}, \pi). \quad (2)$$

We also account for the overhead power consumption at the processor level (e.g., for interference in shared caches) and the node level (e.g., for disk drives, fans, and memory). Overhead power provides a constant power that is not affected by DVFS, thus the greater execution times resulting from low-power P-states can result in greater energy consumption than when just executing the task as fast as possible in P-state P_0. We assume multicore processors can deactivate when all of their cores have finished their assigned workload, and compute nodes are able to shut down when all of their multicore processors are deactivated. We assume deactivated processors and deactivated compute nodes consume negligible power. For our static resource allocation problem with independent tasks, nodes do not have idle time as nodes are active when processing tasks then deactivate when finished with their assigned workload.

Let F_{ij} be the maximum expected completion time among cores in multicore processor j on node i, and σ^2_{ij} be the associated variance. Also, let F_i be the maximum expected completion time among multiprocessors in node i, and σ^2_i be the associated variance. Let ω_i^{MP} be the power overhead for each multicore processor in compute node i, and ω_i^{node} be the power overhead for compute node i. We calculate the expected energy required to process the entire workload across all compute nodes, denoted ζ, as

$$\zeta = \sum_{i=1}^{N} \left(F_i * \omega_i^{node} + \sum_{j=1}^{n_i} (F_{ij} * \omega_i^{MP}) + \sum_{j=1}^{n_i} \sum_{k=1}^{n_{ij}} \sum_{\forall \pi \in P} S^{\pi}_{ijk} \right). \quad (3)$$

The variance of the energy required to process the entire workload, denoted γ, is calculated similarly to ζ, with the exceptions of using the associated variances (σ^2_i, σ^2_{ij}, and Var^{π}_{ijk}) instead of the expected values (F_i, F_{ij}, and S^{π}_{ijk}) and squaring the constants.

The distribution of the energy used to process the workload can be expressed as $\mathcal{N}(\zeta, \gamma)$. Given an energy budget of Δ, we can compute the probability that $\mathcal{N}(\zeta, \gamma)$ is less than Δ (i.e., $\mathbb{P}(\mathcal{N}(\zeta, \gamma) \leq \Delta)$) by converting $\mathcal{N}(\zeta, \gamma)$ to its associated cumulative density function (**cdf**). We denote this probability as ϕ, and we require resource allocations to meet an energy constraint of $\phi \geq \eta$. The values for Δ and η are set by the system administrator.

3.4 Robustness Calculation

We define a "robust" resource allocation as one that can mitigate the impact of uncertain task execution times on our performance objective of finishing all tasks by deadline δ. The calculation of the completion time of core k when using a stochastic model for task execution times is performed by taking the sum of the random variables for each task assigned to core k. The sum of two independent normally distributed random variables α and β produces a normally distributed random variable with its mean being the sum of the means of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ and the variance being the sum of the variances of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

For core *ijk*, the expected finishing time ($\boldsymbol{F_{ijk}}$) and variance ($\boldsymbol{\sigma^2_{ijk}}$) are

$$\boldsymbol{F_{ijk}} = \sum_{\forall t^x_{ijk} \in T_{ijk}} \boldsymbol{\mu}(\boldsymbol{t^x_{ijk}}, \mathbf{PS}(\boldsymbol{t^x_{ijk}})) \quad (4) \text{ and } \boldsymbol{\sigma^2_{ijk}} = \sum_{\forall t^x_{ijk} \in T^x_{ijk}} \boldsymbol{V}(\boldsymbol{t^x_{ijk}}, \mathbf{PS}(\boldsymbol{t^x_{ijk}})). \quad (5)$$

Thus, the completion time distribution of *ijk* can be expressed as $\boldsymbol{\mathcal{N}(F_{ijk}, \sigma^2_{ijk})}$. Given deadline $\boldsymbol{\delta}$, we can compute the probability that $\boldsymbol{\mathcal{N}(F_{ijk}, \sigma^2_{ijk})}$ is less than $\boldsymbol{\delta}$ (i.e., $\mathbb{P}(\boldsymbol{\mathcal{N}(F_{ijk}, \sigma^2_{ijk})} \leq \delta)$) by converting $\boldsymbol{\mathcal{N}(F_{ijk}, \sigma^2_{ijk})}$ to its associated cdf. We define the overall *system robustness*, denoted $\boldsymbol{\Psi}$, as the minimum probability across all cores: $\boldsymbol{\Psi} = \min_{\forall i \in N}(\min_{\forall j \in n_i}(\min_{\forall k \in n_{ij}} \mathbb{P}(\boldsymbol{\mathcal{N}(F_{ijk}, \sigma^2_{ijk})} \leq \delta)))$.

3.5 Combining Performance Metrics

In this study, we use a penalized objective function to incorporate the energy constraint into our optimization techniques. The idea behind a penalized objective function is to reduce the objective function value of infeasible solutions based on the solution's distance from feasibility, but still allow infeasible solutions to be considered when searching for an optimal feasible solution. A *solution* is a complete mapping of tasks to machines. Recall that $\boldsymbol{\phi}$ is the probability that the energy consumption of a resource allocation is less than $\boldsymbol{\Delta}$ and a solution is considered feasible if $\boldsymbol{\phi \geq \eta}$. The distance from feasibility for a solution, denoted $\boldsymbol{d}$, is the difference of $\boldsymbol{\phi}$ from $\boldsymbol{\eta}$: $\boldsymbol{d = \eta - \phi}$.

When $\boldsymbol{d > 0}$, it indicates that $\boldsymbol{\phi < \eta}$, and we penalize the objective function by subtracting a weighted value of the distance from feasibility. Our penalized objective function, denoted $\boldsymbol{\psi}$, is: $\boldsymbol{\psi = \Psi - c * d}$, where $\boldsymbol{c}$ indicates a constant used to control how strongly the constraint will be enforced. A high value of the coefficient $\boldsymbol{c}$ can restrict exploration, however $\boldsymbol{c}$ must be large enough that a feasible solution is found. A value of $\boldsymbol{d \leq 0}$ indicates the energy constraint has been met and we do not need to penalize the solution, and our objective function becomes: $\boldsymbol{\psi = \Psi}$.

4 Heuristics

4.1 Overview

In this section we describe two heuristics: Min-Min Balance and our Tabu Search. Due to lack of space, we omit descriptions for genetic and memetic algorithms, although these are considered in the experiments we conduct.

4.2 Min-Min Balance

Inspired by the greedy iterative maximization heuristic proposed by [14], we create the Min-Min Balance heuristic. Min-Min Balance starts with an initial solution created using the well-known Min-Min Completion Time (Min-Min CT) heuristic [4, 8, 12] and tries to improve the solution using greedy modifications.

The *min-robustness core*, denoted $\boldsymbol{core_{minR}}$, refers to the core that has the least probability of finishing its assigned workload by the deadline, that is, the core that determines the robustness metric of the solution. The *max-robustness core*, denoted $\boldsymbol{core_{maxR}}$, refers to the core that has the greatest probability of finishing its workload by the deadline. Min-Min Balance starts by generating an initial mapping using Min-Min CT with all tasks assigned in $\boldsymbol{P_4}$ (the lowest energy P-state) to ensure the energy constraint is met. The solution is then modified using two steps that reassign tasks and P-states, keeping moves that result in a greater robustness value without violating the energy constraint. The first step iteratively reassigns arbitrary tasks from $\boldsymbol{core_{minR}}$ to $\boldsymbol{core_{maxR}}$ with the goal of increasing robustness by balancing the workload until no tasks from $\boldsymbol{core_{minR}}$ can be reassigned without improving the solution. After the first step, the second step attempts to raise robustness by changing the P-states of tasks on $\boldsymbol{core_{minR}}$ to lower-numbered (i.e., better performing) P-states until no tasks on $\boldsymbol{core_{minR}}$ can change P-states without improving the solution.

4.3 Tabu Search

Overview: The distinguishing feature of Tabu Search is its exploitation of memory, generally through the use of a *Tabu List*. We use a Tabu List to store regions of the search space that have been searched and should not be searched again. Our implementation of Tabu Search combines intelligent local search ("short hops") with the global search ("long hops") in an attempt to find a globally optimal solution.

Local search is performed using three short-hop operators: (1) *task swap*, (2) *task reassignment*, and (3) *P-state reassignment*. One short-hop consists of one iteration of all three operators. Long-hops are performed when local search terminates, with the purpose of jumping to a new neighborhood in the search space, while avoiding areas already searched. After each long-hop, short-hops are again performed to locally search the region near the long-hop solution. The Tabu List stores unmodified long-hops (i.e., starting solutions) that indicate neighborhoods that have been searched before, and may not be searched again. A new solution generated by a long-hop must differ from any solution in the Tabu List by 25%, otherwise a new long-hop is generated. The heuristic terminates after a given number of long-hops are performed.

To evaluate potential task to node allocations we make use of a $\underline{m}$ean $\underline{e}$xecution $\underline{t}$ime rank matrix (**MET rank matrix**) that contains the rank of each heterogeneous node for each task, based on mean execution times. That is, for a given task, the nodes are ranked by how fast the nodes can execute the task (e.g., if node i can execute task t faster than node j, node i is given a better rank for task t). Let the rank of task t on node i be $\mathbf{rank}(\boldsymbol{t_i})$. When comparing the rank of any two tasks A and B on node i, task A is ranked lower (better) than task B if $\text{rank}(\boldsymbol{A_i})$ is less than $\text{rank}(\boldsymbol{B_i})$.

Long-hops: The initial solution (first long-hop) in our heuristic is generated using a Min-Min CT allocation with all tasks running on cores in P-state $\boldsymbol{P_4}$ (to help ensure the energy constraint is met). Subsequent long-hop solutions are generated by first unmapping an arbitrary 25% of tasks, and then reassigning the tasks randomly (i.e., to random cores with random P-states) or heuristically. We try both approaches: one that randomly generates new solutions (*random long-hops*) and one that uses Min-Min CT in $\boldsymbol{P_4}$ to generate new solutions (*Min-Min long-hops*). The three short-hop operators are now described in more detail.

Task Swap: The goal of the *task swap* short-hop operator is to swap tasks that are assigned to poorly *ranked* nodes to better ranked nodes, a move that can potentially improve both robustness and energy consumption. We first choose an arbitrary core k and create a task list consisting of all tasks assigned to core ijk, recalling that the notation for such a task list is $\boldsymbol{T_{ijk}}$. $\boldsymbol{T_{ijk}}$ is sorted in descending order by the rank of each task for node i (e.g., the worst-ranked task is first). We select the first task in the task list, denoted $\boldsymbol{task_A}$, and find the best-ranked node for the task, denoted $\boldsymbol{node_{best}}$. Within $\boldsymbol{node_{best}}$, an arbitrary core z is chosen. The task from core z that has the best rank for node i, $\boldsymbol{task_B}$, is selected for swap. The core assignments for $\boldsymbol{task_A}$ and $\boldsymbol{task_B}$ are swapped, and the best P-state combination (according to the objective function $\boldsymbol{\psi}$) is found and assigned to the tasks. If the solution improves (higher $\boldsymbol{\psi}$), the swap is kept and *task swap* ends. Otherwise, the swap is not kept, $\boldsymbol{task_A}$ is removed from $\boldsymbol{T_{ijk}}$, and *task swap* repeats until the solution improves or $\boldsymbol{T_{ijk}}$ is empty.

Task Reassignment: It can also be useful to transfer tasks from one core to another instead of swapping task assignments, thus we implement a *task reassignment* operator to transfer tasks. The goal of *task reassignment* is to improve robustness ($\boldsymbol{\Psi}$) by removing tasks from the min-robustness core. We start by choosing the min-robustness core (core k) and create a task list consisting of all tasks assigned to the min-robustness core ($\boldsymbol{T_{ijk}}$). $\boldsymbol{T_{ijk}}$ is sorted in descending order by the rank of each task for node i. We select the first task in the task list ($\boldsymbol{task_A}$), and find the best-ranked node for the task ($\boldsymbol{node_{best}}$). Within $\boldsymbol{node_{best}}$, the max-robustness core is selected as the target core (core z), and $\boldsymbol{task_A}$ is assigned to core z. The best P-state (according to $\boldsymbol{\psi}$) is found to run core z in when executing $\boldsymbol{task_A}$. If the solution improves (higher $\boldsymbol{\psi}$), the new assignment is kept and *task reassignment* ends. Otherwise, the new assignment is not kept, task A is removed from $\boldsymbol{T_{ijk}}$, and *task reassignment* repeats until the solution improves or $\boldsymbol{T_{ijk}}$ is empty.

P-state Reassignment: Depending on whether or not the energy constraint has been met, *P-state reassignment* increases or decreases the P-states of tasks to either reduce energy consumption or improve robustness. If the energy constraint is satisfied (i.e., $\boldsymbol{\phi} \geq \boldsymbol{\eta}$), the min-robustness core (core k) is chosen, and a task list is generated consisting of all tasks assigned to the min-robustness core ($\boldsymbol{T_{ijk}}$). A task is chosen arbitrarily from the task list ($\boldsymbol{task_A}$), and the P-state of the task is decreased by 1 if not already currently assigned to execute in $\boldsymbol{P_0}$. If the energy constraint has not been met, the max-robustness core (core k) is chosen, and a task list is generated consisting of all tasks assigned to the max-robustness core ($\boldsymbol{T_{ijk}}$). A task is chosen arbitrarily from the task list ($\boldsymbol{task_A}$), and the P-state of the task is increased by 1 if not currently assigned to execute in $\boldsymbol{P_4}$. If the solution improves, the new P-state is kept and *P-state reassignment* ends. Otherwise, the new P-state is not kept, task A is removed from $\boldsymbol{T_{ijk}}$, and *P-state reassignment* repeats until the solution improves or $\boldsymbol{T_{ijk}}$ is empty.

5 Results

The cluster we simulate consists of 250 compute nodes (N), where the number of multicore processors in a node can vary from one to four ($\boldsymbol{n_i}$), and the number of cores in a multicore processor can vary from two to sixteen ($\boldsymbol{n_{ij}}$), giving a total of 613 multicore processors and 5,430 cores. We conducted 100 distinct simulation trials, with the means and variances of the task execution times varying among trials. These trials simulate numerous diverse heterogeneous workload/system environments. The figures presented in this section show data collected over the 100 simulation trials by displaying the mean values and 95% confidence interval error bars around the mean. The workload of each trial consisted of 40,000 tasks. The mean and variance values of the task execution times for each P-state in each node were generated using the Coefficient of Variation (COV) method [1] and a scaling procedure.

Power consumption values of cores for each node in each P-state ($\boldsymbol{\rho_i^{(\pi)}}$) were generated by sampling a normal distribution for P-states $\boldsymbol{P_4}$ and $\boldsymbol{P_0}$ and interpolating the intermediate states using a quadratic curve. Power overhead values ($\boldsymbol{\omega_i^{node}}$ and $\boldsymbol{\omega_i^{MP}}$) were generated by sampling a uniform distribution with bounds chosen such that the total average overhead power comprised approximately 30% of the total power consumed by the system. We have simulated two different-sized heterogeneous platforms to demonstrate the efficacy of our heuristics. Our primary contribution is not to determine a universally applicable set of parameters, but rather show how parameters for resource allocation heuristics can be calibrated for any given platform through our simulation experiments.

To show the benefits and weaknesses of each local search operator in our Tabu Search heuristic, we tested each operator's performance separately when improving a Min-Min CT $\boldsymbol{P_4}$ solution over time. Figure 1 shows a comparison of each operator separately, as well as all operators working together when improving the Min-Min CT $\boldsymbol{P_4}$ solution, with repeated applications of each operator over the execution time specified on the x-axis. We observed that the improvement when applying the *task reassignment* and *P-state reassignment* operators saturates quickly, whereas the *task swap* operator continues to improve the solution slowly. *Task reassignment* and *P-state reassignment* take greedy approaches by only considering the min-robustness core for task and P-state reassignments, giving strong initial performance but causing the operators to reach local maxima quickly. The *task swap* operator is able to swap assignments on all cores, taking a less greedy approach that allows a broader search. Local search performs the best when all operators are used, thus all operators are used in our Tabu Search heuristic when comparing with other heuristics.

Figure 2a shows a comparison of Tabu Search when using random long-hops versus Min-Min long-hops. Random long-hops can potentially search more of the solution space at the expense of generating poor initial solutions, whereas

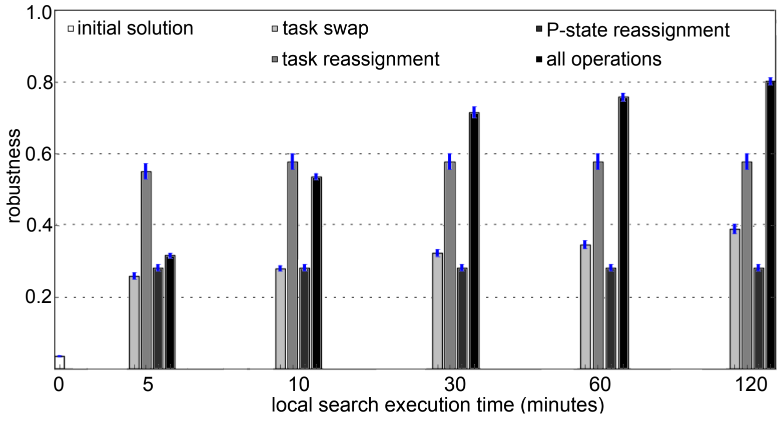
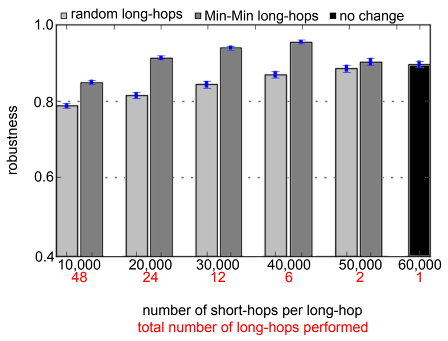


Fig. 1. Local search operator comparison. The figure shows how each operator can improve the robustness of a Min-Min CT $\boldsymbol{P_4}$ solution over time.

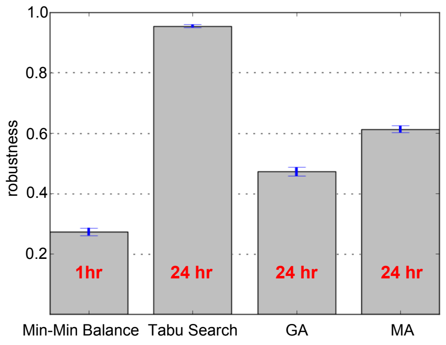
Min-Min long-hops can generate good solutions but limit the search space. Results are shown using varying numbers of long-hop and short-hop iterations to try and understand the tradeoffs between global search and local search, with the number of iterations of each chosen such that the total combined heuristic execution times were the same for the different iteration values. In our environment, we can see that using Min-Min long-hops in our Tabu Search provided significantly better robustness values than using random long-hops, as random long-hops generate poor initial solutions. One conclusion from Figure 2a is that our local search is not improved by global hops for our problem domain when the long-hops are performed randomly, however, global hops become beneficial when performed intelligently using Min-Min.

Figure 2b shows the results obtained from the Min-Min Balance and Tabu Search heuristics, as well as genetic and memetic algorithms (GA and MA) adapted for this environment. The GA is similar to that from [2], and we created the MA by executing the three local search operators from Tabu on offspring chromosomes before they are added to the general population of a GA. The numbers inside the bars indicate the heuristic execution times. The superior performance of Tabu Search and the MA compared to the GA and Min-Min Balance within the computation times given further demonstrate the significance of the local search operations.

We experimented with the heuristics on a smaller system consisting of 25 heterogeneous nodes, 178 total cores, and 4,000 tasks. For that system we observed similar relative performance among the heuristics (as in Fig. 2b), with the Tabu Search and MA outperforming the GA and Min-Min Balance when Tabu Search, GA, and MA are given 24 hours to execute. Though we observed similar relative performances for the two different-sized systems, we leave scalability analysis for future work.



(a)



(b)

Fig. 2. Comparison of (a) random long-hops and Min-Min long-hops and (b) robustness of Min-Min Balance, Tabu Search, GA, MA. All solutions meet the energy constraint.

6 Conclusions

In this paper, we study the problem of statically allocating a bag-of-tasks to a heterogeneous computing system. We proposed new local search techniques that use knowledge of the problem and model to maximize robustness under an energy constraint. We also evaluated and compared Tabu Search short-hop and long-hop techniques, and then combined the local and global search techniques to create a new Tabu Search heuristic. These local search techniques can be adapted to other models and problem domains to enhance Tabu Search, GAs, and other search heuristics with the performance of our Tabu Search heuristic and MA demonstrating the significance of the local search operators in the resource allocation problem domain.

Acknowledgments: The authors thank T. Hansen, M. Amini Salehi, J. Potter, and G. Pfister for their valuable comments. This research was supported by NSF grants CNS-0905399 and CCF-1302693, and by the CSU Abell Endowment. This research used the CSU ISTeC Cray System supported by NSF grant CNS-0923386.

References

1. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D.: Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang J. Science and Engineering, Special 50th Anniversary Issue 3(3), 195–207 (Nov 2000)
2. Apodaca, J., Young, D., Briceño, L., Smith, J., Pasricha, S., Maciejewski, A.A., Siegel, H.J., Bahirat, S., Khemka, B., Ramirez, A., Zou, Y.: Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment. In: Int'l Conf. on Computer Systems and Applications (AICCSA '11). pp. 22–31 (Dec 2011)

3. Boloni, L., Marinescu, D.: Robust scheduling of metaprograms. J. of Scheduling 5(5), 395–412 (Sep 2002)
4. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L., Freund, R.F., Hensgen, D., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel and Distributed Computing 61(6), 810–837 (June 2001)
5. Chang, P.C., Wu, I.W., Shann, J.J., Chung, C.P.: ETAHM: an energy-aware task allocation algorithm for heterogeneous multiprocessor. In: Design Automation Conf. (DAC '08). pp. 776–779 (2008)
6. Cirne, W., Brasileiro, F., Sauv, J., Andrade, N., Paranhos, D., Santos-neto, E., Medeiros, R., Gr, F.C.: Grid computing for bag of tasks applications. In: IFIP Conf. on E-Commerce, E-Business and E-Government (Sept 2003)
7. Hewlett-Packard, Intel, Microsoft, Phoenix Technologies, and Toshiba: Advanced Configuration and Power Interface Specification (2011), rev. 5.0, http://www.acpi.info
8. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. Assoc. Comput. Mach. 24(2), 280–289 (Apr 1977)
9. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Int'l Symposium on High Performance Distributed Computing (HPDC '08). pp. 97–108 (June 2008)
10. Koomey, J.: Growth in data center electricity use 2005 to 2010. Tech. rep., Analytics Press (2011), http://www.analyticspress.com/datacenters.html
11. Lee, Y.C., Zomaya, A.Y.: Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In: IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGRID '09). pp. 92–99 (May 2009)
12. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel and Distributed Computing 59(2), 107–131 (Nov 1999)
13. Shestak, V., Smith, J., Maciejewski, A.A., Siegel, H.J.: Stochastic robustness metric and its use for static resource allocations. J. Parallel and Distributed Computing 68(8), 1157–1173 (Aug 2008)
14. Sugavanam, P., Siegel, H.J., Maciejewski, A.A., Oltikar, M., Mehta, A., Pichel, R., Horiuchi, A., Shestak, V., Al-Otaibi, M., Krishnamurthy, Y., Ali, S., Zhang, J., Aydin, M., Lee, P., Guru, K., Raskey, M., Pippin, A.: Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. J. Parallel and Distributed Computing 67(4), 400–416 (Apr 2007)
15. Xian, C., Lu, Y.H., Li, Z.: Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 27(8), 1467–1478 (Aug 2008)