2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress

Online Resource Management in Thermal and Energy Constrained Heterogeneous High Performance Computing

Mark A. Oxley[*], Sudeep Pasricha[*†], Anthony A. Maciejewski[*], Howard Jay Siegel[*†], and Patrick J. Burns[‡]

[*]Department of Electrical and Computer Engineering
[†]Department of Computer Science
[‡]Vice President for Information Technology
Colorado State University, Fort Collins, Colorado 80523, USA
{mark.oxley,sudeep,aam,hj,patrick.burns}@colostate.edu

***Abstract*—Operators of high-performance computing (HPC) facilities face conflicting trade-offs between the operating temperature of the facility, reliability of compute nodes, energy costs, and computing performance. Intelligent management of the HPC facility typically involves taking a proactive approach by predicting the thermal implications of allocating tasks to different cores around the facility. This offers the benefit of operating the HPC facility at a hotter CRAC temperature while avoiding hotspots. However, such an approach can be a time-consuming process that requires complicated air flow models to be calculated for every mapping decision. We propose a framework in which offline analysis is used to assist an online resource manager by predicting the thermal implications of mapping a given workload. The goal is to maximize the reward earned from completing tasks by their individual deadlines throughout the day, while adhering to a daily energy budget and temperature threshold constraints. We show that our proposed techniques can earn significantly greater reward than traditional load balancing and thermal management schemes.**

***Index Terms*—heterogeneous computing; resource management; thermal-aware computing; energy-aware computing; HPC; DVFS**

I. Introduction

The push to exascale computing is largely prohibited by electricity consumption. For example, the highest-performing supercomputer according to the Top500 list is the Tianhe-2 system that at 33.86 petaFLOPS has a peak power consumption of 17,808 kW, which would cost approximately $17 million per year in electricity using the average cost of electricity in commercial sectors in the U.S. [15, 19]. This prohibitively high cost of energy for an exascale system makes energy-aware management of HPC systems of paramount importance.

The cooling infrastructures in HPC systems consume a significant portion of overall energy, and as such managing resources in a thermal-aware manner is an extremely important aspect of energy-efficient HPC system operation. Providing more cooling than necessary is typical, however, such an approach comes with a higher-than-necessary cost of energy consumption by the cooling infrastructure. One simple solution to reduce the cooling energy cost is to increase the temperature of the facility by raising the thermostat temperature of the computer room air conditioning (CRAC) units to a reasonable temperature under normal conditions (i.e., an average workload), and then relying on compute nodes to throttle themselves to lower processing speeds using dynamic voltage and frequency scaling (DVFS) if the workload becomes high and temperature sensors detect overheating. This approach, however, causes unexpected decreases in performance due to throttling that can result in missed task deadlines. Using thermal models to predict temperatures within the HPC facility allows for proactive resource management techniques to understand the thermal implications of allocating tasks to different cores around the facility. The challenge is that temperature prediction requires complicated and time-consuming airflow models to be calculated for every mapping decision or CRAC thermostat setting. In an online resource management environment, allocation decisions need to be made quickly to start task execution as soon as possible.

The goal of our proposed online resource management framework is to assign dynamically arriving tasks to execute on cores such that the collective reward earned from completing those tasks by their deadline is maximized over a period of time (e.g., a day), within an allowable energy budget allotted over that period of time. We assume there is control over (a) task-to-core mappings, (b) DVFS in cores to allow for performance state (P-state) changes, (c) CRAC thermostat settings, and (d) the perforated floor vent openings where cold supply air from the CRAC enters the HPC facility. We propose a novel resource management framework that uses a database of offline generated solutions, called *templates*, to assist an online resource manager in making thermal-aware decisions in real-time.

In summary, we make the following novel contributions:

- An offline thermal and power-aware optimization formulation for generating templates that considers control of a heterogeneous HPC system and its cooling system. To the best of our knowledge, this is the first work to consider floor vent opening decisions in HPC resource management.
- An online resource management framework that intelligently chooses templates based on the current state of the data center to assist our proposed greedy mapping heuristic in assigning dynamically arriving tasks to cores and P-states. The templates aid the online resource manager in making fast power and thermal-aware mapping decisions with the goal of maximizing reward earned within an energy budget constraint.
- Analysis and comparison of our offline-assisted online resource management scheme under different workload environments and energy budget values with three online mapping schemes and two thermal management strategies from prior work.

II. System Model

A. Overview

We study a heterogeneous HPC facility that is configured in a hot aisle/cold aisle manner (Fig. 1), such as Hewlett Packard's data center research facility in Fort Collins [7]. As demonstrated in Fig. 1, air is supplied from the CRAC units to a cold aisle through perforated floor tiles that face the inlets of the compute nodes. The compute nodes consume power and expel hot air through the opposite end to a hot aisle. The CRAC units draw the hot air from the hot aisles to then cool it. We assume the perforated floor tiles can be in the closed, partially open, or fully open positions.

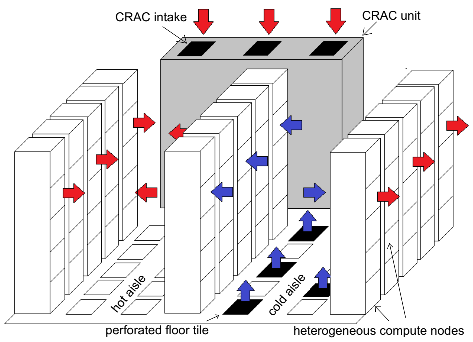

Fig. 1. Data center aisle configuration.

B. Compute Nodes

The computing system is composed of a total of N *compute nodes*, where compute nodes can be one of NT heterogeneous *compute node types*. The compute node type of compute node j is denoted NT_j , where nodes within the same type are identical, meaning they contain the same number of cores and have the same performance and power characteristics. The cores within a node are homogeneous, and we assume the cores can be individually configured to different P-states that offer a tradeoff between power consumption and performance [8]. We denote NC as the *total number of cores* in the HPC system, NC_j as the *number of cores within compute node* j, and CT_k as the *node type to which core k belongs*.

C. Workload

The workload consists of dynamically arriving tasks that can be one of T *task types*. A task type i is defined by its deadline relative to its arrival time (D_i) and the reward earned for completing tasks of that type by the deadline (R_i). The reward earned is representative of a priority level given to different task types, where tasks with high reward are more important for the system to complete. We denote the task type of a task i as TT_i.

Because the system is heterogeneous, tasks of the same type can have different execution times when executed on different node types as well as different P-states. We assume that we know the *estimated time to compute (ETC)* of any task type i on a core k of node type j in P-state $PS_{i,k}$, denoted $ETC_{i,j,PS_{i,k}}$. In an actual system, this can be approximated using historical, experimental, or analytical techniques [9, 12].

D. Power and Energy Models

1) Compute Node Power: We consider the idle power consumption of a compute node (e.g., from main memory, disks, fans) in addition to the non-idle power consumption when the CPU cores are active. The power consumed by cores is a function of the task-type being executed and the P-state in which the core is executing the task. Let P_j^{idle} be the *idle power consumption of compute node* j, let $APC_{i,j,PS_{i,k}}$ be the average power consumed by a core k in a node of type j executing tasks of type i in P-state $PS_{i,k}$. We use these variables to calculate total energy consumption in Section II-D3.

2) CRAC Unit Power: The power consumed at a CRAC unit is a function of the heat removed and the Coefficient of Performance (CoP) of the CRAC unit [14]. Let NCR be the total *number of CRAC units* in the HPC facility, $TC_i^{in}(t)$ be the *inlet temperature of CRAC unit* i at time t, $TC_i^{out}(t)$ be the *outlet temperature of CRAC unit* i at time t, ρ be the *density of air*, C be the *specific heat capacity of air*, and AFC_i be the *air flow rate of CRAC unit* i. The *power consumed by CRAC unit* i at time t, $P_i^{CRAC}(t)$, is calculated as [10]

$$P_i^{CRAC}(t) = \frac{\rho \cdot C \cdot AFC_i \cdot (TC_i^{in}(t) - TC_i^{out}(t))}{CoP(TC_i^{out}(t))}. \tag{1}$$

3) Total Energy Consumption: The energy consumption of a core is the sum of energy consumed for all tasks that are executed on the core over the desired time interval t_p. The *estimated energy consumed* by a core k of type CT_k when executing a task of type i in P-state $PS_{i,k}$, is $EEC_{i,CT_k,PS_{i,k}}$, which is the product of the execution time and average power consumption, $EEC_{i,CT_k,PS_{i,k}} = ETC_{i,CT_k,PS_{i,k}} \cdot APC_{i,CT_k,PS_{i,k}}$.

We assume the idle power of a compute node is constant throughout the day and the *idle energy* consumed by a compute node j over a period of t_p, E_j^{idle}, is $E_j^{idle} = \int_0^{t_p} P_j^{idle} dt = P_j^{idle} \cdot t_p$. The energy consumed by a CRAC unit i is the integral of its power consumption over t_p time. That is, the *CRAC energy consumed* by CRAC unit i, E_i^{CRAC}, is $E_i^{CRAC} = \int_0^{t_p} P_i^{CRAC} dt$.

The *total energy consumed* by the compute and cooling systems, E^{total}, is the sum of core energy, idle node energy, and cooling energy. If $ET_k^{t_p}$ represents the set of *executed tasks* on core k over t_p time, then we calculate E^{total} as

$$\begin{aligned} E^{total} = & \sum_{k=1}^{NC} \sum_{i \in ET_k^{t_p}} EEC_{i,CT_k,PS_{i,k}} \\ & + \sum_{j=1}^{N} E_j^{idle} + \sum_{z=1}^{NCR} E_z^{CRAC}. \end{aligned} \tag{2}$$

E. Transient Thermal Model

The transient thermal model [10] is focused around a collection of *temporal contribution curves* ($c_{i,j}(t)$) and weighting factors $w_{i,j}$. The $c_{i,j}(t)$ curves intuitively represent how long it takes for node (or CRAC) j to experience a temperature change generated by node (or CRAC) i. The $w_{i,j}$ factors represent how much of the heat generated by node i is realized at node j. Using these values, in addition to *outlet temperatures* (T_i^{out}) of all nodes in the facility, we can calculate the *inlet temperature* of any node j at time t, $T_j^{in}(t)$, using Eqn. 5 from [10].

We can calculate the *instantaneous power of node* j at a time t, $PN_j(t)$, by summing the idle power of the node j with the APC values of all cores within node j that are executing tasks at time t. The power consumed by a compute node is dissipated as heat, and the outlet temperature at compute node j at time t is a function of the inlet temperature, the power consumed, and the *air flow rate of the node* AFN_j, calculated using Eqn. 2 from [17]. The outlet temperature of a CRAC unit is assumed to be the value to which the CRAC thermostat is set.

If a thermal violation occurs (i.e., a node's outlet exceeds threshold temperature $T^{threshold}$), all cores within that node are throttled to their highest-numbered (lowest power) P-state until cores have finished executing their current tasks.

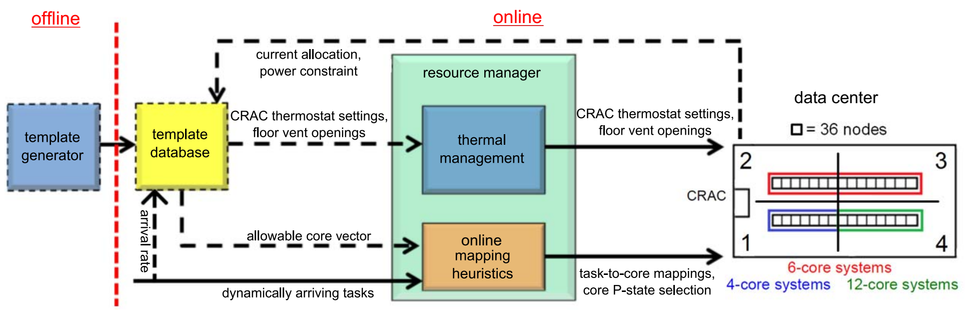
offline
online
template generator
template database
current allocation, power constraint
resource manager
CRAC thermostat settings, floor vent openings
thermal management
CRAC thermostat settings, floor vent openings
data center
= 36 nodes
CRAC
arrival rate
allowable core vector
online mapping heuristics
dynamically arriving tasks
task-to-core mappings, core P-state selection
6-core systems
4-core systems
12-core systems

Fig. 2. Block diagram of the resource management framework. Note that the specific data center shown is for illustrative purposes only, and our framework can be applied to any data center configuration.

F. Steady-state Thermal Model

For our offline template generation, we assume an HPC facility in the steady-state. Steady-state temperatures are calculated in a similar manner as transient temperatures, except without the temporal contribution ($c_{i,j}(t)$) curves. Let $\mathbf{T^{out}}$ and $\mathbf{T^{in}}$ be the vectors of outlet and inlet temperatures of CRAC units and compute nodes. Also, let $\mathbf{W}$ be the matrix of $w_{i,j}$ values that represent the percentage of heat transferred from CRAC unit or node i to CRAC unit or node j. We can then calculate steady-state temperatures by solving the linear combination $\mathbf{T^{in}} = \mathbf{WT^{out}}$ [17].

III. Problem Statement

The objective of our resource manager is to maximize the reward earned over a period of time t_p (e.g., a day) for completing tasks by their individual deadlines. If the set of *tasks that have completed execution by their deadline by time* t_p is TCD_{t_p}, then the goal is to maximize *reward earned by the system* over that period, $R_{system}(t_p)$, calculated as $R_{system}(t_p) = \sum_{i \in TCD_{t_p}} R_{TT_i}$.

The system is constrained by both energy consumption and node outlet temperatures. Over the t_p period of time, the total energy consumption of the facility is not to exceed an energy budget of γ. Reward ceases to be gained when the total energy consumption of the facility over time period t_p exceeds the energy budget, i.e., when $E^{total} \geq \gamma$. This is because in our model, we assume that the facility will discontinue operation until the next period of time.

IV. Proposed Resource Manager

A. Overview

A block diagram of our offline-assisted online thermal and energy aware resource manager is shown in Fig. 2. Dynamically arriving tasks enter the system, and at a *mapping event* the resource manager has to decide the following: (1) which cores within the HPC system to execute those tasks, and (2) the P-states that those selected cores are configured to when executing the assigned tasks. At a *thermal management event*, the resource manager decides (3) CRAC thermostat settings, and (4) which floor vents to close, partially open, or fully open. Dashed lines indicate optional connections between components when using templates for assisting the online resource manager. The mapping event interval time (denoted t_{map}) is set to 60 seconds, and the thermal management event interval time (denoted $t_{thermal}$) is set to 300 seconds in this study.

Template generation is performed offline to create a number of templates in a database that each reflect a steady-state solution for different dynamic states of the HPC facility. We generate a number of templates that vary in three parameters (inputs): (a) newly arrived workload arrival rate, (b) number of cores already executing tasks in each of the quadrants of the HPC facility, and (c) power constraint. The quadrants of the data center we study are shown in Fig. 2. During runtime, our proposed framework chooses a template based on values of those parameters to obtain the values to set the CRAC thermostats, the floor vent opening combinations, and the *allowable core vector* (*ACV*), used by the proposed online mapping heuristic (Section IV-D).

B. Template Input Parameter Definitions

Parameter 1: The first parameter, *newly arrived workload arrival rate*, is calculated as the sum of all arrival rates of the T task types over a considered interval (either the mapping event interval or thermal management interval), weighted by their execution times. We denote this parameter $\lambda^{weighted}_{arriving}(t)$. First, if we denote NCT^{total}_j as the *total number of cores that are of node type* j, then the average execution time for a task type i across all NT node types in the middle P-state for cores within that node type (P^{mid}_j), ETC^{avg}_i, is

$$ETC^{avg}_i = \frac{\sum_{j=1}^{NT} (NCT^{total}_j \cdot ETC_{i,j,P^{mid}_j})}{\sum_{j=1}^{NT} NCT^{total}_j}. \tag{3}$$

The middle P-state is the P-state halfway between the highest-power and lowest-power P-state, and is used in these offline estimations to represent an "average" P-state value. The online mapping heuristics actually select P-states that cores are in when executing various tasks.

The average execution time across all T task types, ETC^{avg}, is then calculated as

$$ETC^{avg} = \frac{\sum_{i=1}^{T} ETC^{avg}_i}{T}. \tag{4}$$

We normalize these values with the average execution time across all T task types using

$$ETC^{normalized}_i = \frac{ETC^{avg}_i}{ETC^{avg}}. \tag{5}$$

If $\lambda_i(t)$ is the arrival rate for tasks of type i over a given time interval t, our *newly arrived workload arrival rate* parameter

($\lambda^{weighted}_{arriving}(t)$) is the sum of weighted arrival rates over all T task types. Intuitively, $\lambda^{weighted}_{arriving}(t)$ is an average amount of "work" arriving to the system that is represented as one parameter. We calculate $\lambda^{weighted}_{arriving}$ by averaging the arrival rates of all task types in the system, weighted by their execution times (as a greater execution time implies more work to be done). $\lambda^{weighted}_{arriving}(t)$ is calculated as $\lambda^{weighted}_{arriving}(t) = \sum_{i=1}^{T} ETC^{normalized}_{i} \cdot \lambda_i(t)$.

Parameter 2: The *number of cores already executing tasks in quadrant q*, $NC^{executing}_{q}$, is the number of cores that are currently executing tasks assigned to them by the dynamic resource manager in a quadrant q from prior mapping events (for a visual example of quadrants, please see Fig. 2). We use the $\lambda^{weighted}_{arriving}(t)$ and $NC^{executing}_{q}$ parameters as estimations of the current state of the HPC facility when the dynamic scheduler is choosing a template.

Parameter 3: The last parameter, the power constraint (denoted P_{const}), is the allotted power the system can use in the steady-state optimization problem.

Because the number of templates to be generated is exponential with the number of input parameters, we limit the total number of parameters in this work to six ($\lambda^{weighted}_{arriving}(t)$, $NC^{executing}_{1}$, $NC^{executing}_{2}$, $NC^{executing}_{3}$, $NC^{executing}_{4}$, and P_{const}). However, the same concepts could be used for a larger number of input parameters (e.g., a finer discretization of a facility than quadrants, or parameters that give exactly *which* cores are executing tasks).

C. Offline Template Generation

The template generation technique tries to find an allocation that matches the service rate of the system with the estimated weighted arrival rate $\lambda^{weighted}_{arriving}(t)$ summed with an estimation of the work that already exists in the system ($\lambda^{weighted}_{existing}(t)$) at a mapping event or thermal management event. The *average execution rate* for a core in node j across all T task types, ER^{avg}_{j}, is

$$ER^{avg}_{j} = \frac{\sum_{i=1}^{T} \left(\frac{1}{ETC_{i,NT_j,P^{mid}_j}}\right)}{T}. \tag{6}$$

Therefore, if $NC^{allowable}_{j}$ represents the number of cores in node j allowed to execute tasks, the service rate of an allocation in this steady-state environment, $\mu_{allocation}$, is $\mu_{allocation} = \sum_{j=1}^{N} NC^{allowable}_{j} \cdot ER^{avg}_{j}$.

Because we formulate this steady-state optimization problem as a non-linear program, we wish to avoid integer decision variables ($NC^{allowable}_{j}$) that create a complex mixed-integer non-linear program, so we introduce a continuous "utilization" decision variable U_j that represents the utilization of node j, and later round this to an integer-valued number of cores, i.e., $\lceil U_j \cdot NC_j \rceil = NC^{allowable}_{j}$. Therefore, we calculate the service rate of a given allocation when using a continuous variable, $\mu^{continuous}_{allocation}$, as $\mu^{continuous}_{allocation} = \sum_{j=1}^{N} U_j \cdot NC_j \cdot ER^{avg}_{j}$.

We assume the steady-state optimization problem uses an "average" workload, and because power consumption of compute nodes is a function of the task types assigned to cores on that node, we take an average of that power across all task types. That is, we assume the *core power consumption* of a core in node j, P^{core}_{j}, in our steady-state formulation is

$$P^{core}_{j} = \frac{\sum_{i=1}^{T} APC_{i,NT_j,P^{mid}_j}}{T}. \tag{7}$$

Therefore, the estimation for total power consumption of the HPC facility in the steady-state, $P^{facility}_{total}$, is

$$P^{facility}_{total} = \sum_{j=1}^{N} (P^{idle}_{j} + U_j \cdot NC_j \cdot P^{core}_{j}) + \sum_{z=1}^{NCR} P^{CRAC}_{z}. \tag{8}$$

Because the template generator is not given information regarding exactly *which* cores are executing tasks by the online resource manager, the assumption is made that the existing work within a quadrant (represented by $NC^{executing}_{q}$) is evenly divided among all nodes within that quadrant. That is, if NC^{total}_{q} is the total number of cores in quadrant q, then each node j in quadrant q has a U_j value of at least $U^q = \frac{NC^{executing}_{q}}{NC^{total}_{q}}$, i.e., U^q represents the fraction of total cores allocated in quadrant q that were already assigned tasks to execute. If NN^q is the *number of nodes in quadrant q*, then the estimated amount of work that exists in the system ($\lambda^{weighted}_{existing}(t)$), is calculated as $\lambda^{weighted}_{existing}(t) = \sum_{q=1}^{4} \sum_{j=1}^{NN^q} U^q \cdot NC_j \cdot ER^{avg}_{j}$.

The decision variables for the steady-state optimization problem are the utilizations of compute nodes, U_j, and the CRAC thermostat temperatures, TC^{out}_{i}. The following equation shows the optimization problem for the steady-state offline optimization problem:

$$\text{maximize} \frac{\mu_{allocation}}{P^{facility}_{total}} \tag{9}$$

subject to

1) $\mu_{allocation} \leq \lambda^{weighted}_{arriving}(t) + \lambda^{weighted}_{existing}(t)$
2) $U^q \leq U_j \leq 1, \forall j \in q, \forall q$
3) $P^{total}_{facility} \leq P_{const}$
4) $T^{out}_{j} \leq T^{threshold}, \forall j$

The objective is to maximize the amount of work performed per unit power, where the decision variable U_j affects both $\mu_{allocation}$ and $P^{facility}_{total}$, and decision variable TC^{out}_{i} affects $P^{facility}_{total}$ as the thermostat temperature of the CRAC units contributes to the power consumption of the facility. Constraint 1 guarantees that the allocated service rate does not exceed the amount of work to be done (newly arriving and existing). Constraint 2 guarantees that the utilization of each node (representative of the number of cores that are allowed to be active in that node) is greater than that already assigned to that node (represented by U^q), and less than 1.0 (as a node cannot have more cores allowed to have tasks assigned to it than the total number of cores that exist in that node). Constraint 3 guarantees the power constraint. Constraint 4 guarantees the thermal constraints (a node's outlet temperature cannot exceed $T^{threshold}$). Rounding is performed (using a ceiling function) to obtain discrete $NC^{allowable}_{j}$ values from the continuous U_j values.

Floor vent openings for each of the quadrants are not considered a decision variable in the presented optimization problem, as they are integer variables (either closed, partially open, or fully open), and accurately relaxing these to continuous variables is out of the scope of this paper. Thus, when generating the templates, we perform the above optimization when considering distinct thermal models for all possible floor vent opening combinations (closed, partially open, or fully open) in all quadrants and record the best one to store in the database.

D. *Online Mapping Heuristic*

At the start of a mapping event, our greedy deadline-aware heuristic (Alg. 1) drops all tasks in the unmapped batch that would not be able to make their deadlines even when executed on their fastest node type in the fastest P-state (line 1). Next, the *ACV* is obtained. The *ACV* is a vector composed of the $NC_j^{allowable}$ values, where element j of the *ACV* is the number of cores in node j that are allowed to have tasks mapped to them. If using templates to assist in resource allocation, the *ACV* is obtained from the template in the template database that most-closely represents the parameters given in Section IV-B (line 2). When not using templates (i.e., using a comparison thermal management technique from Section V), the *ACV* consists of all cores within the HPC facility (line 3). From the *ACV*, a subset of cores called *idle allowable cores* is obtained, which is a set of cores not currently executing tasks in the *ACV* (line 4). That is, the idle allowable cores in a node j is the number of cores within that node that are both allowed to have tasks mapped to them and are currently not executing any tasks. The idle allowable cores in a node j are found as follows: if a node j has less cores that are currently executing tasks than what the value of ACV_j is, then the difference between those values is how many more cores within node j are idle allowable cores, and are allowed to have tasks mapped to them.

The unmapped batch of tasks is then sorted in descending order by their reward earned if completed by their deadline (line 5). The algorithm then iteratively assigns tasks to the core (from the set of idle allowable cores) and P-state combinations that result in the lowest energy while still meeting the deadline until there are no tasks left to map, or until there are no idle allowable cores to map tasks (lines 6-10). If n is the number of idle allowable cores at a mapping event, and m is the number of tasks in the unmapped batch at a mapping event, then the complexity of the heuristic is approximately $\mathcal{O}(n^2 + mlogm)$ if $m > n$, or $\mathcal{O}(mn)$ if $n > m$.

Algorithm 1 Pseudo-code for greedy deadline-aware heuristic

```
1. drop tasks that cannot meet deadline
2. if using templates then obtain ACV from template database
3. else set ACV to all cores in system
4. obtain idle allowable core set
5. sort unmapped batch of tasks in descending order by reward
6. while unmapped batch is not empty and idle allowable core set is not empty
7.    select first task in unmapped batch
8.    find core/P-state combination from set of idle allowable
      cores that gives lowest energy and meets deadline
9.    assign task to that core and P-state
10.   remove task from unmapped batch and remove core from idle allowable core set
11. end while
```

V. Comparison Techniques

A. *Overcooling and Throttling Thermal Management Strategies*

The thermal management techniques presented in this section provide the CRAC thermostat temperatures ($TC^{out}(i,t)$). The *overcool* strategy simply sets the CRAC thermostat to a low constant temperature (22°C). The *throttling* strategy sets the CRAC thermostat to a relatively high constant temperature (28°C), and resorts to throttling overheating nodes if needed. We assume that when using the *overcool* or *throttling* approaches, the *ACV* (cores that are allowed to have tasks actually mapped to them) is composed of all cores in the system. The floor vents are set to the fully-opened configuration for both *overcool* and *throttling* strategies.

B. *Online Mapping Heuristics*

For all techniques in this subsection, cores are configured to execute assigned tasks in P-state P0, which represents the highest performance and highest power P-state.

1) Consolidation: At the beginning of a mapping event, the *consolidation* technique (similar in concept to techniques found in [13, 21]) assigns unmapped tasks in an arbitrary order to a core within the system that is not currently executing any task. Ties (where multiple cores are not executing any tasks) are solved by assigning the task to the first core in the list of all cores in the HPC system (*NC*). In this work, cores are ordered in the list starting with all cores in quadrant 1 (from left-to-right by rack in Fig. 2, and from top node to bottom node in each rack), then quadrant 2, etc.

2) Load Balancing by Node: At the beginning of a mapping event, the *load balancing by node* (*LBBN*) technique assigns unmapped tasks in an arbitrary order to the node in the system that is currently executing the least number of tasks (e.g., FIRSTAVAILABLE from [1]). When a node is chosen for assignment, a random core within that node is assigned to execute the task. Ties (where multiple nodes have the same least number of cores executing tasks) are solved by assigning the task to the first node in the list of all N nodes.

3) Load Balancing by Rack: At the beginning of a mapping event, the *load balancing by rack* (*LBBR*) technique assigns unmapped tasks in an arbitrary order to the rack (as defined as a vertical cabinet containing multiple nodes, e.g., represented by a black square in Fig. 2) that is currently executing the least number of tasks (similar in concept to UniformWorkload [14]). When a rack is chosen for assignment, a random core within a random node within the selected rack is assigned to execute the task. Ties (where multiple racks have the same least number of cores executing tasks) are solved by assigning the task to the first rack in the list of all racks.

VI. Simulation Setup

A. *Overview*

We consider a heterogeneous platform composed of one CRAC unit, and 30 server racks with 36 compute nodes per rack (a total of 1,080 compute nodes), as shown in Fig. 2. Each node is one of three node types (see Table I). The threshold temperature for compute nodes was set to 33°C, which is on the high end of ASHRAE's data center temperature guidelines [3]. The CoP for a CRAC unit is a function of its outlet temperature, τ, given by $CoP(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458$ [14]. The air flow rate of each CRAC unit is set to 26.1 m^3/s.

TABLE I
Node types used in simulations

node type	Lenovo TS140	HP Z600	HP Z820
total # of nodes	270	540	270
# of cores in node	4	6	12
# of P-states	16	9	16
air flow rate (m^3/s)	0.0284	0.0519	0.0566

B. Workload

We study bursty and sinusoidal workload arrival patterns. When assuming the bursty workload pattern, tasks arrive to the system according to a Poisson process at intervals that switch periodically throughout the day between a λ_{low} arrival rate set to 3.0, and a λ_{high} arrival rate set to 9.0. We also consider a sinusoidal arrival rate pattern that uses data from the MetaCentrum2 workload trace [5]. The task type of a task is randomly chosen based on a uniform distribution.

The execution time and power characteristics for different task-types are from the PARSEC benchmark suite [20], and corresponding ETC and APC matrix entries for each benchmark on each node type in each P-state are obtained by measuring the execution times and average power consumption of those benchmarks on each of the machines and P-states from Table I. The deadlines for tasks of a given type were set to the average execution time for that task type across all node types in the P_{mid} P-state (see Equation 3). The reward values earned for completing tasks of a given type by their individual deadlines are linearly proportional to their execution times (i.e., tasks with longer execution times have higher reward values).

C. Template Generation

We generated a range of templates for the parameters listed in Section IV-B for our *templates* technique. For the newly arrived workload parameter ($\lambda_{arriving}^{weighted}(t)$), we used the values [2.2, 6.6, 11.0, 15.4, 19.8], as these values evenly divide the range of [0,22], and 22 is approximately the maximum service rate capability for the system in Table I. The number of cores in quadrant 1 currently executing tasks ($NC_1^{executing}$) is discretized using the values [180, 540, 900] that splits the 1,080 total number of cores in that quadrant. Similarly, the number of cores executing tasks in quadrants 2 and 3 ($NC_2^{executing}$ and $NC_3^{executing}$) are discretized using the values [200, 605, 1010, 1415], and the number of cores executing tasks in quadrant 4 ($NC_q^{executing}$) is discretized into [180, 540, 900, 1260, 1620, 1980, 2340, 2700, 3060]. Last, the power constraint parameter (P_{const}) is discretized into [150, 170, 190, 210, 230] kilowatts.

VII. Results

In this section we present results, discussion, and analysis of our proposed offline-assisted online resource management framework, and compare it with the *consolidation*, *LBBN*, and *LBBR* mapping heuristics that use either *overcooling* or *throttling* thermal management approaches. The bar graphs showing results for the bursty arrival rate pattern represent the averages of 12 trials, with each trial varying in the actual arrival times of tasks and the task types of arriving tasks. The error bars are the 95% confidence intervals around the mean of those 12 trials.

The results presented in Fig. 3 compare the *consolidation*, *LBBN*, *LBBR*, and *greedy* mapping techniques in combination with the *overcooling*, *throttling* thermal management techniques, as well as the greedy technique with the *templates* thermal management technique that represents our offline-assisted online resource management framework. Fig. 3 (a) and (c) show the reward earned by the different techniques for the (a) bursty workload arrival pattern, and (c) sinusoidal workload arrival pattern. Similarly, Fig. 3 (b) and (d) show the energy consumption of those techniques for the bursty and sinusoidal arrival rate patterns, respectively. These results are compared across a wide range of different daily energy budget values from 8000 MJ to 16000 MJ that represent a range from a highly constrained system to a less constrained system. For instance, *consolidation* using the *overcooling* thermal management technique is only able to process tasks for approximately half of the day when given an energy budget of only 8000 MJ. The system uses approximately 22000 MJ per day when all cores in the facility are executing the highest-powered task in P-state P0, and the cooling system is set in the *overcooling* management scheme. However, for energy budgets above 16000 MJ, results were the same as the results for 16000 MJ for the techniques and arrival rates simulated in this study.

We can observe in Fig. 3 (a) that our proposed greedy heuristic in combination with templates provides a significant benefit in reward earned at lower energy budget levels (less than 14000 MJ). At such low energy budget levels, the system is unable to remain activated for the entire day, regardless of the mapping heuristic or thermal management technique chosen, i.e., the energy in the budget is exhausted before the day ends and the facility is essentially deactivated, not allowing any more tasks to complete. However, the greedy online mapping heuristic is able to exploit DVFS to save energy, and when using templates is able to reduce cooling energy consumed (compared to *overcooling*), and therefore is able to complete more tasks by their deadline than the other techniques.

We can also analyze some other interesting trends in Fig. 3 (a) by examining other techniques. First, we can observe that when using the *consolidation* online mapping technique, the *overcooling* thermal management technique significantly outperforms the *throttling* thermal management technique. However, when using *LBBN* and *LBBR*, the *throttling* technique earns significantly more reward than *overcooling* technique. This is because *consolidation* maps most tasks spatially close to each other (i.e., in the same quadrant), whereas *LBBN* and *LBBR* spatially distribute the tasks in the facility. This means that when using the *throttling* thermal management technique (i.e., operating the facility at a hotter temperature and relying on the throttling mechanisms to prevent overheating), the *consolidation* mapping technique packs all tasks in a corner of the facility and causes hotspots in that area, invoking the throttling mechanisms almost constantly and causing a large number of missed deadlines. The *throttling* technique works well for *LBBN* and *LBBR* because tasks are more spatially distributed, therefore causing fewer hotspots (and thus not invoking the throttling mechanisms), while still being able to operate the facility at a hotter temperature and save on cooling energy to use more of the energy budget for computing. The *overcooling* thermal management technique performs better for the *consolidation* technique than *LBBN* or *LBBR* because the nodes that the *consolidation* technique consolidates the workload on (primarily those in quadrant 1 in Fig. 2) happen to also, in general, consume less energy than the other nodes.

Another interesting observation is that, when given a large amount of energy budget (e.g., 16000 MJ), Fig. 3 (a) shows that using the greedy online mapping technique in combination with the *overcooling* thermal management technique still slightly outperforms the comparison heuristics that use *overcooling*, even though those techniques assign all tasks in P-state P0 and

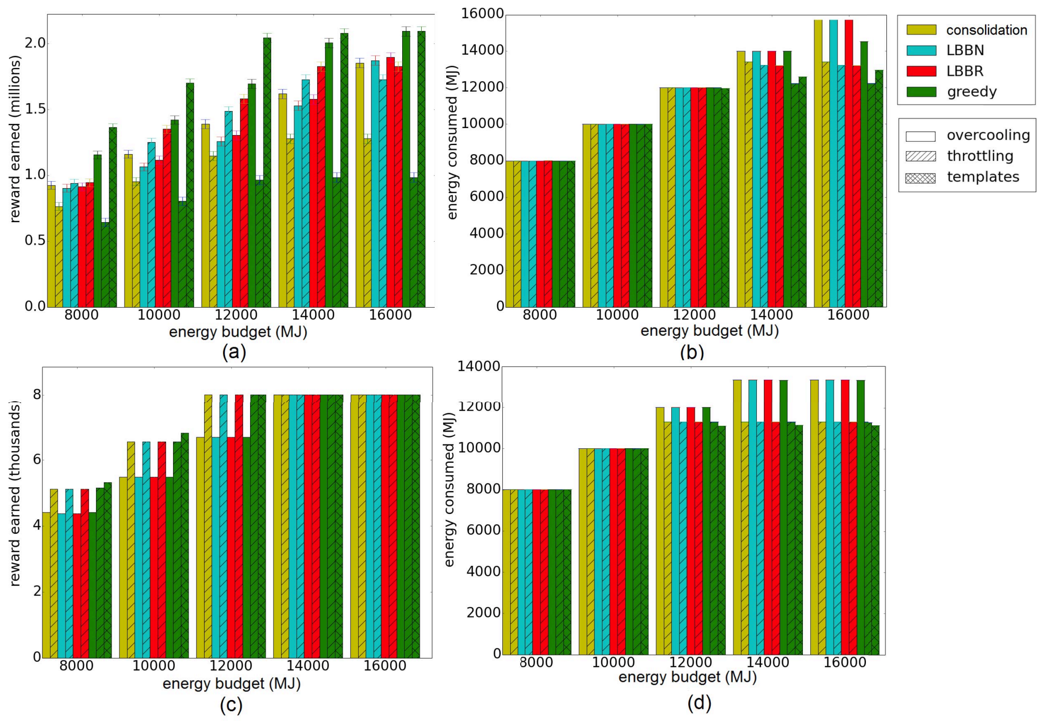

Fig. 3. Compares reward earned by techniques for the (a) bursty, and (c) sinusoidal workload arrival patterns. Similarly, (b) and (d) display energy consumed by those workload arrival patterns. All results in this figure are shown for the four online mapping techniques (consolidation, LBBN, LBBR, and greedy), and three thermal management strategies (overcooling, throttling, templates).

the system has a large enough energy budget to finish the whole day. This is because the comparison techniques do not exploit any form of heterogeneity. Thus, even when executing a task in P0, a task may still miss its deadline when executed on a low-performance node if the task had to wait in the unmapped queue for awhile until a mapping event was triggered. One last observation from Fig. 3 (a) is that the *throttling* thermal management technique performs very poorly when paired with the *greedy* online mapping technique. This is because the *greedy* mapping technique assigns tasks to execute in the lowest-energy P-state that still hits the deadline. As a result, if any throttling occurs from overheating, tasks will very likely miss their deadlines.

Fig. 3 (c) displays the comparison of reward earned by techniques across several energy budget levels when using the sinusoidal arrival rate pattern. It is important to recall that this experiment used the real trace from the MetaCentrum2 system from the Parallel Workload Archive [5], and even though the computing systems are similar in size, the trace has a significantly lower arrival rate for tasks compared to the *bursty* arrival rate pattern that was generated. As such, much less reward can be earned (as evidenced by the y-axis of Fig. 3 (c) in the magnitude of thousands compared to Fig. 3 (a) that shows reward in millions). Therefore, given enough energy, all tasks can easily complete by their deadlines, as shown by the reward earned by all techniques when given an energy budget of at least 14000 MJ. Also, the *throttling* technique outperforms the *overcooling* technique for all mapping heuristics, and this is because even though the facility is at a higher temperature for the *throttling* approach (28°C), the small number of tasks being executed using this workload trace does not cause the compute nodes to generate much heat and trigger the throttling mechanisms.

We examine the energy consumption of the facility when using the different techniques for the bursty workload arrival rate pattern in Fig. 3 (b), and the sinusoidal pattern in (d). At energy budgets of only 8,000 MJ and 10,000 MJ, all techniques use all of the available energy budget before the day ends. However we can see in Fig. 3 (a) and (c) that our *greedy* online mapping heuristic in combination with the *templates* thermal management technique makes the best use of the limited amount of energy and earns significantly greater reward than the others. When using the *overcooling* thermal management approach, the facility requires nearly 16000 MJ of energy to last the entire day without deactivating due to exceeding the energy budget. We can also observe in Fig. 3 (b), (d), and (f) that when using our offline-assisted online mapping framework (i.e., the greedy mapping technique in combination with the templates thermal management approach), energy can be saved compared to the *overcooling* technique when given a larger energy budget (e.g., 16,000 MJ). The *templates* technique uses slightly more energy than *throttling* at higher energy budget levels, but the reward earned using the *throttling* technique is approximately half that of when using the *templates* technique.

VIII. Related Work

Thermal management of HPC systems and data centers is an important and active research topic. The first explorations into studying thermal management in HPC facilities focused on modeling temperatures and heat. Moore et al. [14] introduced the notion of heat recirculation among compute nodes and proposed a technique that reduces heat recirculation and cooling costs. Due to the inefficiency of using CFD simulations to conduct thermal evaluations, Tang et al. [17] proposed an abstract heat flow model that determines coefficients arranged into a matrix that denote percentage of heat that is transferred from any node in the system to all other nodes in the system. Later, a transient thermal model was proposed by Jonas et al. [10] that also gives temporal information, that is, how the temperatures evolve throughout time rather than assuming steady-state. Our work is different from those works as it focuses on resource management rather than modeling, and proposes online resource management rather than steady-state as in [14].

Several thermal-aware works have been published that focus on reducing cooling power or energy [4, 11, 16]. In [4], CFD data was studied to find that an effective thermal-aware technique to reduce cooling power is to migrate workload from the node with the highest inlet air temperature. The research of [11] minimized cooling energy by identifying "better cooled" areas of the data center and consolidating more virtual machines (VMs) in those areas and allowing other unused servers to be deactivated. In [16], offline resource management techniques, specifically a genetic algorithm and quadratic programming approach, were proposed to minimize cooling energy. The primary difference between our work and [4], [11], and [16] is that we also consider optimization of the *performance* of the HPC system in conjunction with the energy consumption of facility by considering reward earned.

Performance is also considered as a design objective in some thermal-aware works [2, 6, 22]. The research in [2] considers both performance optimization under a power constraint and power optimization under a performance constraint in a steady-state offline framework. Trade-offs between energy cost savings and workload delay can be exploited using the Stochastic Cost Minimization Algorithm technique proposed in [6]. The research in [22] studied power management in an internet data center to minimize power of both compute and cooling systems while meeting quality of service (QoS) constraints on latency for servicing web requests. We differentiate from [2] by considering an online resource management framework rather than offline steady-state. The research in [6] does not consider heterogeneity, DVFS, heat recirculation, floor vent opening control, or CRAC unit control as we do in our work. Our work considers floor vent opening control as well as tasks with different priority levels (reward) and deadlines rather than throughput optimization as in [18]. The problem studied in [22] is different than our work, by minimizing power of both computing and cooling systems under latency constraints in an internet data center environment. We also differ from [22] by considering floor vent opening control and a node power model that is dependent on the type of workload being executed.

IX. Conclusions

We studied the problem of maximizing the reward collected for completing tasks by their deadlines subject to a daily energy budget and thermal constraints for heterogeneous high-performance computing systems. The primary contribution of this research was our proposed offline-assisted online resource management framework designed to solve this problem, where a number of templates are generated offline to assist the online resource manager make thermal-aware decisions based on the incoming workload and state of the HPC facility. We performed extensive analysis and compared our framework with three techniques that use either an *overcooling* approach to thermal management, or operating the facility at a higher temperature and relying on throttling for thermal management. We demonstrated how our framework can greatly increase the reward earned within a daily energy budget by intelligently providing enough cooling to avoid triggering the throttling mechanisms, but operating the facility hot enough to also save on cooling energy, allowing more of the energy budget to be spent on computing.

Acknowledgments

The authors thank Daniel Dauwe and Dylan Machovec for their valuable comments on this work. This research was supported by NSF grants CCF-1302693 and CCF-1252500. This research used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386. We also thank Hewlett Packard for donating servers for this work.

References

[1] Adaptive Computing. Node allocation policies (Moab workload manager), 2015.
[2] A. M. Al-Qawasmeh, S. Pasricha, A. A. Maciejewski, and H. J. Siegel. Power and thermal-aware workload allocation in heterogeneous data centers. *IEEE Trans. Comput.*, 64(2):477–491, 2015.
[3] A. T. Commitee. 2011 Thermal Guidelines for Data Processing Environments - Expanded Data Center Classes and Usage Guidance. Technical report, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc., 2011.
[4] D. Demetriou and H. Khalifi. Thermally aware, energy-based load placement in open-aisle, air-cooled data centers. *J. Electronic Packaging*, 135(3), 2013. 14 pp.
[5] D. G. Feitelson. Parallel workload archive, 2015.
[6] Y. Guo, Y. Gong, Y. Fang, P. P. Khargonekar, and X. Geng. Energy and network aware workload management for sustainable data centers with thermal storage. *IEEE Trans. Parallel and Distributed Systems*, 25(8):2030–2042, 2014.
[7] Hewlett Packard. Hp opens new research facility to advance sustainable data center technologies, 2011.
[8] Hewlett-Packard, Intel, Microsoft, Phoenix Technologies, and Toshiba. *Advanced Configuration and Power Interface Specification*, 2011. Rev. 5.0.
[9] M. A. Iverson, F. Ozgüner, and L. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Trans. Comput.*, 48(12):1374–1379, 1999.
[10] M. Jonas, R. R. Gilbert, J. Ferguson, G. Varsamopoulos, and S. K. S. Gupta. A transient model for data center thermal prediction. In *3rd Int'l Green Comp. Conf. (IGCC '12)*, 2012. 10 pp.
[11] E. K. Lee, H. Viswanathan, and D. Pompili. VMAP: Proactive thermal-aware virtual machine allocation in HPC cloud datacenters. In *19th Int'l Conf. on High Performance Comp. (HiPC '12)*, 2012. 10 pp.
[12] 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 and Distributed Comp.*, 44(1):33–52, 1997.
[13] H. Lin, X. Qi, S. Yang, and S. Midkiff. Workload-driven VM consolidation in cloud data centers. In *29th Int'l Parallel and Distributed Processing Symp. (IPDPS '15)*, pages 207–216, 2015.
[14] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *USENIX Annual Technical Conf. (ATEC '05)*, pages 61–75, 2005.
[15] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer. The TOP 500 list, 2015.
[16] Q. Tang, S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel and Distributed Systems*, 19(11):1458–1472, 2008.
[17] Q. Tang, T. Mukherjee, S. K. Gupta, and P. Cayton. Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters. In *4th Int'l Conf. on Intelligent Sensing and Information Processing (ICISIP '06)*, pages 203–208, 2006.
[18] O. Tuncer, K. Vaidyanathan, K. Gross, and A. K. Coskun. Coolbudget: Data center power budgeting with workload and cooling asymmetry awareness. In *32nd Int'l Conf. Computer Design (ICCD '14)*, pages 497–500, 2014.
[19] United States Energy Information Administration. Electric power monthly, 2015.
[20] P. University. The PARSEC Benchmark Suite, 2015.
[21] H. Viswanathan, E. Lee, I. Rodero, D. Pompili, M. Parashar, and M. Gamell. Energy-aware application-centric VM allocation for HPC workloads. In *25th Int'l Parallel and Distributed Processing Symp. (IPDPS '11)*, pages 890–897, 2011.
[22] J. Yao, H. Guan, J. Luo, L. Rao, and X. Liu. Adaptive power management through thermal aware workload balancing in internet data centers. *IEEE Trans. Parallel and Distributed Systems*, 26(9):2400–2409, 2015.