Value and Energy Aware Adaptive Resource Allocation of Soft Real-time Jobs on Many-core HPC Data Centers

Amit Kumar Singh, Piotr Dziurzanski and Leandro Soares Indrusiak
Department of Computer Science, University of York, Deramore Lane, Heslington, York, YO10 5GH, UK.
Email: {amit.singh, piotr.dziurzanski, leandro.indrusiak}@york.ac.uk

***Abstract*—Modern high performance computing (HPC) data centers consume huge energy to operate them. Therefore, appropriate measures are required to reduce their energy consumption. Existing efforts for such measures focus on consolidation and dynamic voltage and frequency scaling (DVFS). However, most of them do not perform adaptive resource allocation for the executing dependent tasks (or jobs) in order to optimize both value and energy. The value is achieved by completing the execution of a job and it depends on the completion time. A high value is achieved if the job is completed before its deadline; otherwise a lower value. In this paper, we propose an adaptive resource allocation approach that uses design-time profiling results of jobs for efficient allocation and adaptation in order to optimize both value and energy while executing dependent tasks. The profiling results for each job are obtained by exploiting efficient allocation combined with identification of voltage/frequency levels of used system cores and used in adapting to different number of cores based on the monitored execution progress of the job and available cores. Experiments show that the proposed approach enhances the overall value by about 10% when compare to existing approaches while showing reduction in energy consumption and percentage of rejected jobs leading to zero value.**

I. Introduction

Large scale HPC data centers integrate several many-core architectures to enhance their processing capability, but a huge energy is required to operate them [1], [2]. It has been reported that energy consumption of data centers is between 1.1% and 1.5% of the worldwide electricity consumption and the consumption is expected to increase rapidly in future [3]. Further, the power requirements of these systems are increasing rapidly. This stringent increase in power requirement cannot be fulfilled due to physical limitations on the available energy, which has put some financial data centers out of energy, e.g., Morgan Stanley datacenter in 2010 [4]. Thus, it is of paramount importance to minimize energy consumption of these systems during their operation.

In a many-core HPC data center that typically contains several connected servers, jobs arrive at different moments of time and they need to be serviced by allocating them on the available cores on different servers at run-time. In doing so, the value (utility) achieved by servicing the jobs should be maximized while trying to minimize the overall energy consumption during system operation as mentioned earlier. A job may contain a number of dependent/independent tasks or processes. The notion of values (economic or otherwise) of jobs has been introduced to define their importance level [5]. In overload situations where demand for available resources is higher than the supply, such a notion facilitates in deciding to allocate limited resources to the high value jobs and holding the low value jobs for late allocation.

The value of a job can change over time to reflect the impact of the computation over the business processes. Usually, the change in value of a job over time is determined by considering its soft real-time deadline [6], [7]. If the job is completed before the deadline, a high value is achieved; otherwise, a low value is achieved. This also implies that the violation of deadline does not make the computation irrelevant, but reduces its value for the user [5], [8], [9]. Deadlines missed by large margins may result in zero value and thus the computation becomes useless for the user. Further, the energy spent on such computation can be considered as wasted. Therefore, the job request should be rejected if no (zero) value can be obtained by executing it. Consideration of such varying value depending upon the completion time and deadline increases complexity to the allocation process.

For each job, the allocation outcome determines the value to be achieved after completing the job and its energy consumption as well. Additionally, if the platform cores support dynamic voltage and frequency scaling (DVFS) [10], the voltage/frequency (v/f) levels of used cores by the job also govern the value and energy consumption. A lower v/f level represents lower dynamic power consumption level and thus v/f levels of one or more used cores can be adjusted depending upon their workload in order to reduce energy consumption while not violating timing constraints [6]. Thus, to jointly optimize both the metrics (value and energy), efficient allocation along with appropriate v/f levels of used cores need to be identified.

The identification of efficient allocation and v/f levels has been accomplished by performing design-time profiling of the jobs [11]. These profiling results are used to facilitate light-weight run-time resource allocation as the compute intensive part is shifted to design-time. Such allocation approaches have been proven promising to design job-specific-clouds, where the clients (or customers) and their jobs to be submitted for execution are pre-defined, which can be realized from the historical data [11], [12]. However, these approaches do not perform run-time adaptive allocation by monitoring the availability of cores on different servers and execution status of the jobs.

Motivational Example: Fig. 1 shows a motivational example to execute an arrived job on one server of a many-core HPC system (data center) when employing non-adaptive

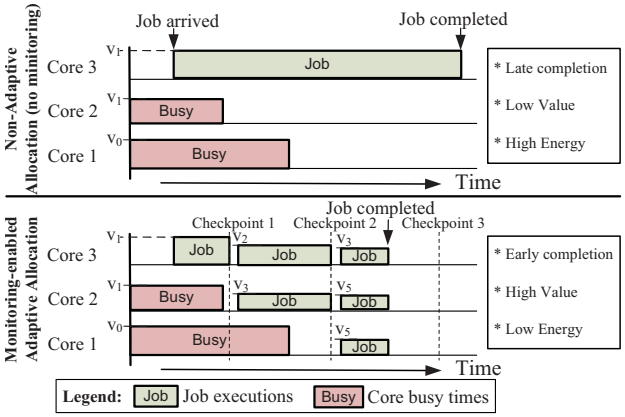

Fig. 1. Motivation example showing non-adaptive and adaptive allocation.

and adaptive allocations. The server contains three cores and two of them are busy, i.e., executing some other jobs, when the *job arrived*. The non-adaptive allocation approaches (top part of Fig. 1) always execute the job on the same allocated core at an identified v/f level (e.g., on core 3 at voltage level V_1) in view of optimizing completion time (determining value) and energy consumption. The operating voltages for different executions are represented by various heights. Since no motoring is performed for the job execution status to check whether it is executing slower of faster than expected (based on average or worst-case execution time) and availability of cores, the operational v/f and number of used cores remain constant. This might lead to *high energy* consumption and *late completion*, resulting in *low value*. **Opportunity:** By employing monitoring [13], the execution status of the jobs and availability of cores over the servers can be made available at different moments of time, e.g., at *checkpoint 1* and *checkpoint 2* (bottom part of Fig. 1). These checkpoints occur periodically at a specified interval. Based on the execution status at the checkpoints, the number of cores to be used by the job and operational v/f levels of cores can be changed (increased or decreased) to jointly optimize the value (depending on completion time) and energy consumption. For example, at *checkpoint 1*, the job is reallocated to two cores from one core and to three cores from two cores at *checkpoint 2* while applying appropriate v/f levels to cores. To perform efficient run-time adaptation, the premier allocations and v/f levels for different number of cores can be identified at design-time in order to use them at run-time [11], [14]. Note that the adaptation step can also migrate the job to some other server of the HPC system that has high availability of cores at current checkpoint. These considerations might lead to early job completion to realize high value, and low energy consumption due to executions at lower operating voltages.

Contribution: In this paper, we propose a monitoring-enabled adaptive resource allocation approach to efficiently execute jobs arriving at different moments of time on a many-core HPC system. The jobs are profiled in advance to identify the efficient allocations and v/f levels from value and energy point of view when using different number of cores. Since jobs are profiled in advance, it is assumed that all the clients (or customers) and their jobs to be submitted for execution in the HPC data center are known in advance. This is true for several data centers as they serve only a fixed set of known customers and such information facilitates to design promising job-specific-clouds [11], [12]. The profiling results are used to perform efficient run-time allocation and adaptation. The monitoring information in terms of jobs execution status and availability of system cores at different moments of time is used as feedback to decide whether allocation and v/f levels of used cores by an executing job should be changed to jointly optimize value and energy. In case the change is beneficial, the proposed approach performs adaptation (migration) on higher or lower number of cores by selecting the appropriate allocation and v/f levels of cores from the profiling results. The adaptation step can perform migration within the same or to a different server and takes such migration overheads into account.

II. Related Work

Existing resource allocation approaches for HPC systems employ various principles to optimize the overall value to be achieved by servicing the arrived jobs. For example, an approach in [15] chooses the highest value job first. This approach might lead to small amount of available resources if a high value job requires a large amount of resources. As a remedy to this problem, the job having maximum value density can be chosen first, where the value density is computed as value divided by the amount of required computational resources [16]. Variants of value density approaches have also been proposed [16]–[18]. Another approach to pre-empt the low value executing jobs in order to assign freed resources to high value arrived jobs is also proposed [19]. However, these approaches do not consider energy consumption optimization and DVFS capable cores. Further, run-time adaptive allocation is not employed.

Energy optimization approaches for HPC data centers have focused mainly on virtual machines (VMs) consolidation and DVFS. In consolidation, VMs with low utilization are placed together on a single host (server) so that other used hosts can be freed to shut them down [20]–[22]. DVFS approaches for HPC data centers are explored to save dynamic energy consumption [6], [11]. The approach of [6] does not consider jobs containing dependent tasks, whereas [11] considers them. For dependent tasks, DVFS approaches from other domains, e.g. embedded systems [14], can be employed, but they do not perform optimization for value. Further, energy can also be optimized by performing energy-aware resource allocation [23]. The approaches considering DVFS and optimizing both the value and energy consumption are recently reported [5], [11], but they do not perform adaptive resource allocation as job execution status and available system cores are not monitored.

There has been some efforts to achieve (monitor) the job execution and system cores status in order to make decisions to perform adaptive resource allocation [13]. However, these efforts do not consider dependent jobs or tasks and do not

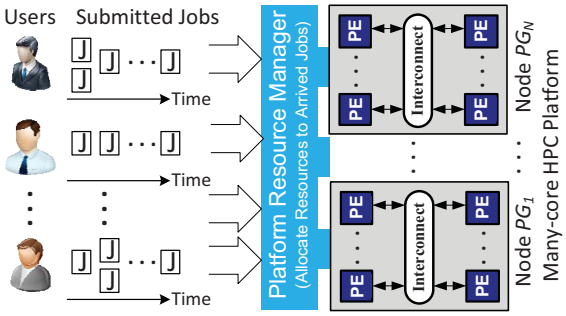

Fig. 2. System model adopted in this paper. A cloud data center containing different nodes (servers) with dedicated cores (PEs) to execute jobs (Js) submitted by multiple users.

exploit DVFS and design-time profiling results. In contrast, our approach considers all the above aspects. The use of profiling results facilitates efficient initial allocation with appropriate v/f levels of used cores and efficient adaptation on lower or higher number of cores based on the monitoring status, resulting in optimized value and energy.

III. System Model and Problem Definition

Our system model is based on typical industrial HPC scenario and presented in Fig. 2. Various *users* submit a set of *jobs* at different moments of time to be executed in the *many-core HPC platform (data center)* . The jobs are submitted to the *platform resource manager* that allocates resources to them. This section provides preliminaries pertaining to the system model along with the problem definition.

A. Job and its Value Curve Model

Each job j is modelled as a directed acyclic graph $TG = (T; E)$, where T is the set of tasks of the job and E is the set of directed edges representing dependencies amongst the tasks. Fig. 3 (a) shows an example job containing 4 tasks (t_1,..,t_4) connected by a set of edges. Each task $t \in T$ is associated with its execution time (*ExecTime*, measured as worst-case execution time (WCET)), when allocated on a core operating at a particular voltage level. Such information can be obtained from previous executions of the tasks. Each edge $e \in E$ represents data that is communicated between the dependent tasks. A job j is also associated with its arrival time AT_j.

Each job is considered to have a soft real-time deadline, implying that the violation of deadline does not make the computation irrelevant, but reduces its value for the user [5], [6], [8]. The value of the job to the user depends upon the completion time and is represented by *value curve*. Fig. 3 (b) shows a typical value curve, where vertical and horizontal axes show the value and completion time, respectively. It indicates that if the job is completed before it's soft deadline, a high value as that of the value at job arrival is achieved. After the deadline, the value curve is a monotonically-decreasing function and trends towards zero with the increasing completion time, as shown in Fig. 3 (b). This implies that the value achieved decreases with increased in the completion time and deadline missed by large margins may result in zero value.

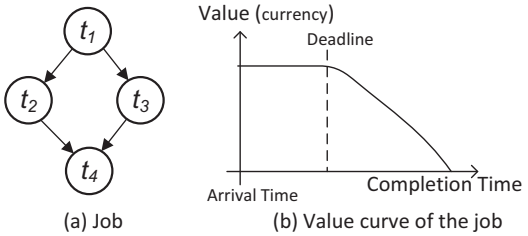

Fig. 3. An example job model and its value curve.

We assume value curve of each job is given and the value curve incorporates the soft real-time deadline by representing the value over the completion time based on the deadline. Such a value curve reflects job's business importance assessed by the end user while following a domain specific economic model. The description of the economic model is orthogonal to our approach and out of scope of this paper.

B. Many-core HPC Platform Model

The HPC platform HP contains a set of nodes $(PG_1, ..., PG_N)$, as shown on the right hand side of Fig. 2. A node (server) n contains a set of homogeneous cores C_n, referred to as processing elements (PEs), which communicate via an interconnect. Each core is assumed to support DVFS [24]. A *platform resource manager* controls access of platform resources and coordinates the execution status of jobs submitted by the users, which facilitates efficient management of resources and incoming requests.

C. Energy Consumption of a Job

The total energy consumption (E_{total}) of a job is computed as the sum of dynamic and static energy:

$$E_{total} = E_{dynamic} + E_{static} \quad (1)$$

The dynamic energy consumption for all the tasks in the job is estimated from equation (2).

$$E_{dynamic} = \sum_{\forall t \in T} (ExecTime[t] \rightarrow c_v) \times (pow \rightarrow c_v)] \quad (2)$$

where $ExecTime[t] \rightarrow c_v$ and $pow \rightarrow c_v$ are the execution time of task t mapped on core c operating at voltage v, and corresponding power consumption, respectively. It is assumed that the power consumption at different operating voltages is known in advance and taken from chip manufacturer's data sheet. E_{static} is computed as the product of overall execution time of the job and static power consumption of the used cores.

D. Problem Definition

To efficiently service the arrived jobs, the target research problem considers the following set of input, constraints and objective.

- **Input:** Workload, i.e., Job set $(j_1, ..., j_M)$, Value curve of each job VC_j incorporating its soft real-time deadline, Arrival time of each job AT_j ($j \in \{1, \ldots, M\}$), Cores of the HPC platform nodes $(PG_1, ..., PG_N)$, Voltage levels $(v_1, ..., v_l)$ supported by each core.

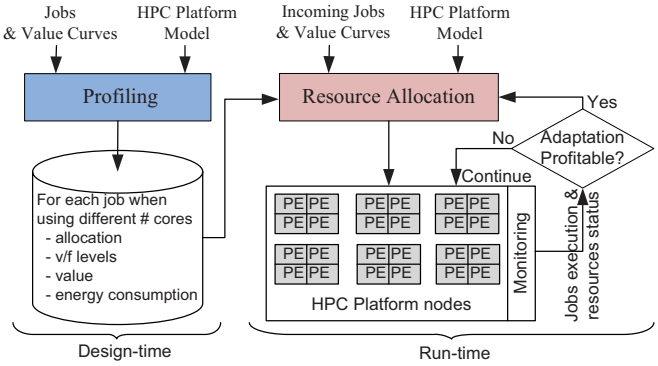

Fig. 4. Monitoring-enabled adaptive resource allocation.

- **Constraints:** Limited resources (cores) on each node of HP.
- **Objective:** Maximize overall value Val_{total} and minimize energy consumption E_{total}.

For an arrived job, first, the allocation process followed by the platform resource manager needs to identify the node to execute the job, tasks-to-cores allocation inside the node, and the v/f levels of the cores executing tasks of the job. Then, depending upon the job execution status and available system cores, the job should be reallocated within the current or to a different node with a new tasks-to-cores allocation and v/f levels. This reallocation (adaptation) process is repeated whenever it is beneficial from value and energy perspective. Since there are several possible allocations (tasks-to-cores assignments) for a job and several voltage scaling (VS) options for each allocation, exploring the complete design space to identify the optimal design in terms of value and energy might not be feasible within acceptable time. Therefore, only efficient allocations and appropriate VS options need to be evaluated. Further, for dependent tasks, applying VS on a core is rather challenging as one needs to capture the VS effect on the execution of dependent tasks allocated on other cores.

IV. Proposed Monitoring-enabled Adaptive Resource Allocation Approach

In contrast to conventional existing efforts that consider only few aspects in the allocation process, our approach considers all the following aspects: *1)* jobs containing dependent tasks, *2)* apply DVFS, *3)* jointly optimize value and energy, *4)* utilize design-time profiling results, and *5)* perform adaptive resource allocation.

An overview of our proposed approach is provided in Fig. 4. The approach performs *design-time profiling* of the jobs obtained from the historical data. At *run-time*, the profiling results are used to perform efficient allocation for the arrived jobs and adaptation (reallocation) for the executing jobs. The decisions for the reallocation are made based on the *jobs execution and resources' availability status*, which are obtained by a monitoring framework. We consider a similar monitoring framework as that of [13]. The details of the *profiling*, *allocation*, and *reallocation* steps are as follows.

A. Design-time Profiling

For each job, the profiling step identifies the *allocation* and *v/f levels* leading to optimized response time (determines *value*) and *energy consumption* when utilizing different amount of computing power in terms of number of cores. The response time should be minimized to optimize value and is calculated as the difference between the end and start time of the job execution after allocating resources to it. To minimize both response time and energy consumption, we consider to minimize the product of response time and energy consumption in order to jointly optimize value and energy. At different number of used cores, since an exhaustive search might not be performed within a limited time, the allocation and v/f levels leading to the minimum product value are identified by employing a genetic algorithm (GA) based evaluation. We chose NSGA-II as the underlying GA to steer the optimization process [25]. The number of cores is varied from one to the number of tasks in the job in order to exploit all the potential parallelism present in the job by assuming that each task can occupy only one core. For each job, the *allocation*, *v/f levels*, *value* corresponding to the response time and *energy consumption* at different number of used cores are stored as the profiling results (Fig. 4).

B. Run-time Resource Allocation and Reallocation

To perform resource allocation and reallocation (adaptation) by using the profiling results, the manager follows Algorithm 1. For all the arrived jobs, the algorithm takes profiling results from the storage, their value curves, and the HPC Platform HP as input and identifies a value and energy optimizing allocation based on the number of available cores at different nodes in the platform. Further, at various checkpoints (monitoring points), the algorithm checks execution status of allocated/executing jobs and availability of resources to identify a new allocation and v/f levels of used cores in order to perform reallocation whenever profitable.

The algorithm checks mainly for three events as follows: *1) any already allocated job(s) finish execution* to update the platform resources (lines 1-3), *2) any job(s) arrive into the platform* to be put into a job queue (lines 4-6), and *3) checkpoint occurs during execution* to try to perform adaptation (reallocation). In case of both the events *1)* and *2)* or any of them, the algorithm tries to perform resource allocation for the queues job(s) having non-zero values (lines 18-28). However, if event *3)* is also detected at the same time, the *adaptation* is tried first (lines 7-17) followed by the *allocation* (lines 18-28). This ensures that the executing jobs are given priority over the queued jobs to be allocated so that value and energy of the executing jobs can be further optimized before doing optimizations for the queued jobs. If such measures are not taken, further optimization opportunity for the executing jobs can be missed, which may lead to lower overall value and higher energy consumption. The details of the *adaptation* and *allocation* steps are as follows.

1) Adaptation (lines 7-17): The adaptation (reallocation) for all the executing jobs is tried at all the checkpoints incurred over the system execution. These checkpoints usually occur at

ALGORITHM 1: Resource Allocation and Adaptation

Input: Incoming Jobs with arrival times, Jobs' profiling results and value curves, HPC Platform HP.
Output: Resource Allocation for Incoming Job and Reallocation for Executing Jobs.

1 **if** *allocated_job(s) finish execution* **then**
2 | Update platform resources;
3 **end**
4 **if** *job(s) arrive* **then**
5 | Put the *job(s)* in *JobQueue*;
6 **end**
7 **if** *checkpoint occurs* **then** `// Try Adaptation`
8 | **for** *count = 0 to nrExecutingJobs* **do**
9 | | Capture $deviations$ of executing and non-adapted jobs;
10 | | Estimate $remainedTime$, $newEndTime$ and $newValueDensity/newEnergyDensity$ of deviated jobs when utilizing different number of available cores in various nodes;
11 | | Select $maxValuePerEnergyDensityJob$, its $Node$, $nrUsedCores$, $new\ v/f\ levels$, $newValue$, $newEnergy$, and $newAllocation$;
12 | | **if** $newValueDensity/newEnergyDensity > valueDensity/energyDensity$ && $reallocationGain > reallocationOverhead$ **then**
13 | | | Adapt (Reallocate) $maxValuePerEnergyDensityJob$ on $nrUsedCores$ cores of selected $Node$ by following $newAllocation$ to perform execution at $new\ v/f\ levels$;
14 | | | Update platform resources;
15 | | **end**
16 | **end**
17 **end**
18 **if** *JobQueue contains job(s) having positive values* **then** `// Try Allocation`
19 | **for** *count = 0 to JobQueue.size()* **do**
20 | | Collect $bids$ from all nodes and select $maxBid$;
21 | | **if** $maxBid > 0$ **then**
22 | | | Compute $value/energy$ estimates of unscheduled jobs when utilizing $maxBid$ cores;
23 | | | Select $maxValuePerEnergyJob$ and its $value$, $energy$, $allocation$, and $v/f\ levels$ from profiling results;
24 | | | Schedule $maxValuePerEnergyJob$ on node having $maxBid$ cores by following the $allocation$ to perform execution at $v/f\ levels$;
25 | | | Update platform resources;
26 | | **end**
27 | **end**
28 **end**

a regular interval, which can be varied. If these checkpoints occur quite often, the reallocation might need to be performed frequently. Since there is an overhead to check the need for reallocation at each checkpoint in terms of time and energy, the frequent checks might delay the jobs completion time and thus overall value might get reduced and energy consumption might get increased considering the fact that energy needs to be dissipated for a longer time. In case of less frequent checkpoints, the instances for profitable reallocations can be missed and it might not be possible to optimize the value and energy for the executing jobs. The effect of the varying number of checkpoints on the considered performance metrics (value and energy) is shown in the next section.

At each checkpoint, to perform reallocation for all the executing jobs, all of them (*count* = 0 to *nrExecutingJobs*, line 8) are tried to be reallocated on a higher or lower number of cores based on the jobs execution status and availability of cores. The jobs execution status is captured from the monitoring framework in terms of their *deviation* from the actual expected progress at the checkpoint. If the deviation is positive, i.e. job execution has exceeded the checkpoint, it is tried to be reallocated on a higher number of cores within the currently allocated node or on a different node with a new allocation and v/f levels. The maximum number of used cores is the number of tasks in the job, which can exploit all the potential parallelism. However, in case of negative deviation, the job is tried to be reallocated on a lower of number of cores on the currently allocated node. The reallocation process continues until all the executing jobs are tried to be reallocated.

To perform reallocation, first, deviations of executing and non-adapted jobs are captured (line 9). The non-adapted jobs are those for whom reallocation has not been performed. Then, for all the deviated jobs, i.e., jobs with positive or negative deviations, remained time to complete ($remainedTime$), new end time ($newEndTime$), and value density divided by energy density ($newValueDensity/newEnergyDensity$) are estimated when using different number of available cores in various nodes (line 10). These entities are estimated as follows.

$$remainedTime = remainedWork \times completionTime_{usedCores} \tag{3}$$

$$newEndTime = progreesedTime + remainedTime \tag{4}$$

$$\frac{newValueDensity}{newEnergyDensity} = \frac{newValue/remainedTime}{newEnergy/remainedTime} \tag{5}$$

where $completionTime_{usedCores}$ is the completion time at the used number of cores and $remainedWork$ is computed by looking the deviation included end time and the deviation at the current checkpoint, and it is normalized with respect to the total work to be done from start to end time of the job. The $progreesedTime$ is the job progressed time by the checkpoint. The selection of value and energy density helps to choose the job leading to maximum value and minimum energy consumption with minimal remaining completion time ($remainedTime$). Thus, a job is selected that will complete soon and lead to high value and low energy consumption. This also provides opportunity to use the released resources by early completion for allocating queued jobs or profitable reallocations.

From all the deviated jobs, the one leading to the maximum value per energy density ($maxValuePerEnergyDensityJob$, computed by Equation 5) is selected along with its node, number of used cores, v/f levels of cores, new value, new energy and new allocation (line 11). The allocation, v/f levels, value and energy consumption are simply selected from the profiling results, which facilitates for fast run-time computations. For the new allocation of $maxValuePerEnergyDensityJob$, if the new value density over energy density is greater than that of the previous allocation and estimated reallocation gain is greater than the reallocation overhead, the job is reallocated to the selected node based on the new allocation to perform

execution at identified new v/f levels. The reallocation overhead consists of time and energy required to find a profitable instance of adaptation and perform reallocation to a higher or lower number of cores within the same executing node or a different node. The overhead to reallocate (migrate) the tasks to a different node is higher than that of the migration within the same node. These overheads are taken into account along with the overhead to set the cores to be used on new v/f levels. Such a reallocation leads to optimized value and energy. After performing reallocation, the platform resources are updated and opportunity for the next reallocation is explored.

2) Allocation (lines 18-28): To perform resource allocation for all valuable queued jobs (i.e., jobs having positive values), all of them (*count* = 0 to *JobQueue.size()*, line 19) are tried to be allocated on the platform resources as along as any core is available. The allocation process ensures that a queued job having zero value at the allocation time is dropped from the queue as no value can be made out of it. The allocation process continues until all the arrived jobs are allocated or dropped due to having zero value while waiting in the job queue.

In the allocation process, first, *bids* (in terms of number of available cores) from different platform nodes are collected, then the maximum bid ($maxBid$) and the corresponding node is selected (line 20). Choosing such a node to use its cores helps to achieve better load balancing amongst nodes and thus better resource utilization. In case more than one nodes have the same amount of bid, any of them is chosen. If the estimate of $maxBid$ is greater than zero ($maxBid > 0$, line 21), i.e., at least one core is available in the platform, the value/energy estimates of jobs utilizing $maxBid$ cores are computed and the job leading to maximum value per energy consumption ($maxValuePerEnergyJob$) is selected to be scheduled to the node having $maxBid$ cores by following the allocation and v/f levels leading to the optimized value and energy. The computation of value/energy for each job considers its value at the allocation time and the exact number of cores to be used by the job computed as minimum between $maxBid$ and the number of cores to be used to achieve maximum value/energy. The platform resources are updated after scheduling each job to have up to date resources' availability information for the next allocation instance. This helps to achieve an accurate and efficient allocation. Similar process is repeated for all the arrived jobs.

V. Experimental Results

The proposed resource allocation approach is implemented in a C++ prototype and integrated with a SystemC functional simulator. As a workload, job models from historical data of an industrial HPC system (data center) at High Performance Computing Center Stuttgart (HLRS) are considered, where the jobs have varying arrival time. To sufficiently stress the platform, we consider all the jobs arriving over a month. To remain close to the reality, it is considered that higher number of jobs arrives in peak times, i.e. weekdays and daytimes as compared to off-peak times, i.e. weekends and night times. Each job contains a set of dependent tasks as described earlier. The number of tasks in the jobs varies from 5 to 10 and tasks execution is in the order of minutes or hours. Further, it is assumed that the value curve of each job is given.

To evaluate our approach for different number of available servers (nodes), varying number of nodes are considered in the HPC platform, where each node contains a total of 10 cores as in modern servers. Further, to evaluate the approach for assorted chip manufacturing technologies that will enable higher number of cores at each node, experiments are performed with varying number of cores at each node while considering a fixed number of nodes. The number of cores is varied such that it covers a broad spectrum of technologies including advanced servers to be available in future. The platform cores are assumed as the cores of Intel Core M processor, which supports 6 v/f levels of operation. The reallocation overhead along with the overhead to set the v/f levels is taken into account. For each job, such overheads are computed and stored in advance in order to use them during run-time reallocation.

In addition to overall value and energy consumption, we also evaluate the percentage of rejected jobs that are removed from the job queue as their value becomes zero before the resources become available to allocate them. The rejected jobs also include jobs achieving zero value after their execution, which can be prevented by employing proper admission control and schedulability analysis. Further, we have also analyzed the effect of number of checkpoints on the value and energy consumption in order to identify the exact number of check points or checkpoint interval leading to optimized value and energy consumption.

Experimental Baselines: We compare results obtained from our approach to those of [15], [14], and [11] as they can be applied to jobs containing dependent tasks. Table I summarizes these approaches and their used abbreviations. The approaches of [15] and [14] do not use profiling results and thus allocations and v/f levels are identified at run-time. The allocations in these approaches are found in a manner such that the load across the used cores is balanced. The approach of [15] optimizes only the value by finding an efficient allocation while keeping the operating v/f levels of the used cores at the highest levels. This approach helps to recognize energy savings by approaches applying DVFS. To employ this approach, the adaptation step (lines 7-17 in Algorithm 1) from our approach is removed and value optimizing allocations are found at run-time. The approach of [14] identifies v/f levels of used cores to optimize only energy consumption. Therefore, it has been extended to optimize both the value and energy for a fair comparison. To employ this approach, the efficient allocations leading to minimal response time (determining value) are found as described earlier and then the v/f levels of the used cores by the greedy algorithm of [14] that fixes v/f levels of cores one-by-one during consecutive iterations. Further, the adaptation step from Algorithm 1 is removed. This approach optimizes value and energy separately in the identifications of allocations and v/f levels, respectively, and is referred to as *ValEnSepOpt*. In [11], the value and energy are optimized jointly by utilizing the profiling results and the approach is referred to as *ValEnJoinOpt*. To employ this approach, the

TABLE I
APPROACHES CONSIDERED FOR COMPARISON

Approaches	Abbreviation	References
Value Optimization	ValOpt	[15]
Value and Energy Separate Optimization	ValEnSepOpt	[14]
Value and Energy Joint Optimization	ValEnJoinOpt	[11]
Value and Energy Adaptive Optimization	ValEnAdaptOpt	Proposed

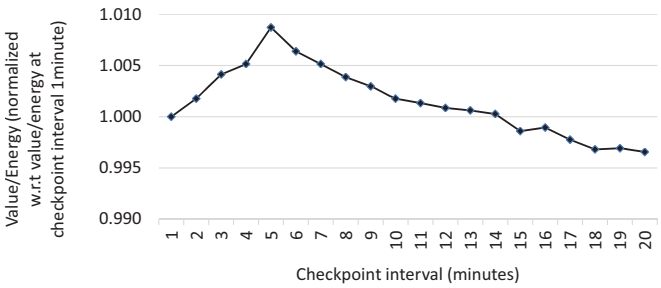

Fig. 5. Value/Energy at different number of checkpoints.

adaptation step is removed from Algorithm 1. In addition to utilizing profiling results, our approach (Algorithm 1) employs adaptation and is referred to as *ValEnAdaptOpt*.

A. Effect of number of checkpoints on value and energy consumption

Fig. 5 shows the value over energy (value/energy) estimates achieved by our approach ValEnAdaptOpt when different number of checkpoints is considered by varying the checkpoint intervals. The shown result considers 3 available nodes in the platform, where each node contains 10 cores. A lower checkpoint interval represents higher number of adaptations, where an adaptation is tried at the regular checkpoint interval. It can be observed that when adaptation is tried at every minute, i.e. checkpoint interval is 1, value/energy estimate is low due to frequent adaptation trials, which incurs high overhead in terms of time and energy and the timing overhead delays the allocation and thus completion (determining value) of queued jobs. The value/energy estimate initially increases with the checkpoint interval as the check-pointing overheads are reduced. At higher values of checkpoint interval, the estimate decreases as opportunities of adaptation are missed for higher number of executing jobs. This indicates that the number of checkpoints should be appropriately chosen mainly based on the job arrival patterns and the same has been considered for all the conducted experiments.

B. Value and energy consumption with varying number of nodes

Fig. 6 shows the influence of the number of available nodes (servers) on the overall value and energy consumption when various approaches are employed. The value and energy results are normalized with respect to (w.r.t.) the value and energy by ValOpt approach at 2 nodes. For our approach, fixed check-pointing intervals of five minutes are considered after analyzing value and energy at various checkpoint intervals as described in the previous subsection. A couple of observations can be made from Fig. 6. *1)* Overall value by all the approaches increases with the number of nodes due to increased

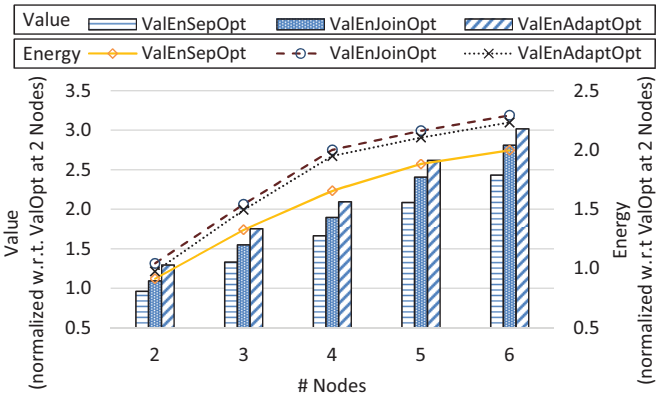

Fig. 6. Value and Energy with varying number of nodes.

processing capability leading to completion of higher number of jobs before their value becomes zero. *2)* ValEnAdaptOpt approach achieves higher overall value than other approaches. This is due to the fact that adaptation leads to early completion of executing jobs and thus higher values for them. Further, earlier completion leaves resources for the queued jobs to be allocated and completed sooner, leading to higher values. *3)* ValEnAdaptOpt performs better than other approaches if both the value and energy metrics are to be jointly optimized as value achieved per unit of energy consumption, i.e. value divided by energy. On an average, ValEnAdaptOpt achieves 9.46% higher value than that of ValEnJoinOpt, which provides better results as compared to other existing approaches.

C. Value and energy consumption with varying number of cores in each node

Fig. 7 shows the overall value and energy consumption when the number of cores at each node is varied from 10 to 20 for a total of 3 considered nodes. The value and energy results are normalized w.r.t. the value and energy by ValOpt approach at 10 cores. A couple of observations can be made from the figure. First, the value by all the approaches increases with the number of cores due to increased processing capability leading to completion of higher number of jobs before their value becomes zero. Second, ValEnAdapt achieves higher overall value than other approaches. Additionally, when both value and energy needs to be jointly optimized as value achieved per unit of energy consumption, i.e. value over energy, ValEnAdapt provides better results as compared to other approaches.

D. Percentage of rejected jobs

Table II shows the rejected jobs (%) at different number of available nodes when various approaches are employed. The average over all the nodes is also shown for all the approaches. It can be observed that, on an average, our proposed approach ValEnAdaptOpt rejects lower number of jobs as compared to the baseline approaches. Our approach has lowest rejection of jobs as the adaptation process completes executing jobs early and freed resources are used by the queued jobs before their values become zero. Thus, lower rejections are achieved. It can also be observed that the rejections are lowered with

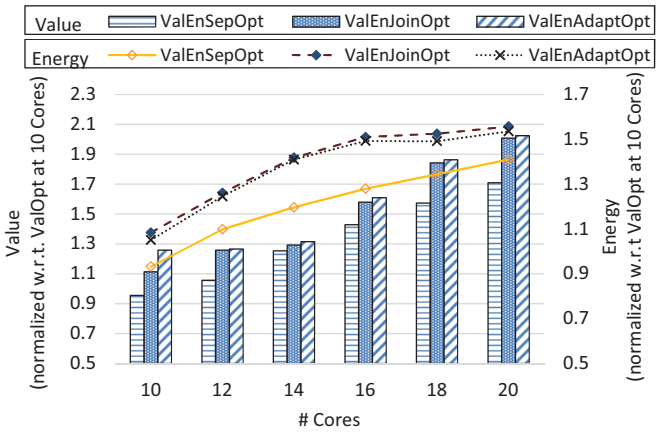

Fig. 7. Value and Energy with varying number of cores at each node.

TABLE II
PERCENTAGE OF REJECTED JOBS AT DIFFERENT NUMBER OF NODES

	ValOpt	ValEnSepOpt	ValEnJoinOpt	ValEnAdaptOpt
2	58.8%	56.2%	48.8%	48.8%
3	41.2%	37.4%	26.2%	25.8%
4	26.2%	22.8%	07.4%	07.2%
5	14.2%	13.6%	00.0%	00.0%
6	08.6%	08.6%	00.0%	00.0%
Average	29.8%	27.7%	16.4%	16.3%

increased number of nodes due to larger number of resources' availability. This metric is important from users' satisfaction point of view as they do not want their jobs to be rejected.

VI. Conclusions and Future Work

We have proposed an adaptive resource allocation approach for HPC data centers. The approach uses design-time profiling results to perform efficient allocation and reallocation. The profiling step combines identification of efficient allocation and appropriate v/f levels to jointly optimize value and energy consumption. It has been shown that efficient allocation and reallocation has led to significant reduction in energy consumption and enhancement in value. In future, we plan to extend our approach to heterogeneous HPC data centers, where servers may contain different types of processing cores.

Acknowledgment

This work is funded by EU FP7 DreamCloud project under grant agreement no. 611411.

References

[1] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole, "Energy-efficient application-aware online provisioning for virtualized clouds and data centers," in *International Green Computing Conference (IGCC)*, 2010, pp. 31–45.

[2] C. Conficoni, A. Bartolini, A. Tilli, G. Tecchiolli, and L. Benini, "Energy-aware cooling for hot-water cooled supercomputers," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2015, pp. 1353–1358.

[3] J. Koomey, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, 2011.

[4] D. J. Brown and C. Reams, "Toward Energy-efficient Computing," *Commun. ACM*, vol. 53, no. 3, pp. 50–58, 2010.

[5] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system," *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, 2015.

[6] R. N. Calheiros and R. Buyya, "Energy-Efficient Scheduling of Urgent Bag-of-Tasks Applications in Clouds Through DVFS," in *Proceedings of IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM)*, 2014, pp. 342–349.

[7] F. Caglar, S. Shekhar, and A. Gokhale, "iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-Based Real-Time Applications," in *IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, June 2014, pp. 48–55.

[8] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-Based Task Service," in *IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2004, pp. 160–169.

[9] K. Chen and P. Muhlethaler, "A Scheduling Algorithm for Tasks Described by Time Value Function," *Real-Time Syst.*, vol. 10, no. 3, pp. 293–312, 1996.

[10] *Intel Core i7 Processor Series Datasheet, Vol. 1*, Intel Corporation, 2010, http://www.intel.com/.

[11] A. K. Singh, P. Dziurzanski, and L. S. Indrusiak, "Value and Energy Optimizing Dynamic Resource Allocation in Many-core HPC Systems," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015.

[12] S. Kim and Y. Kim, "Application-specific Cloud Provisioning Model Using Job Profiles Analysis," in *IEEE International Conference on High Performance Computing and Communication*, June 2012, pp. 360–366.

[13] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong, "Resource monitoring and management with OVIS to enable HPC in cloud computing environments," in *Proceedings of IEEE International Parallel Distributed Processing Symposium (IPDPS)*, 2009, pp. 1–8.

[14] A. K. Singh, A. Das, and A. Kumar, "Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems," in *Proceedings of ACM Design Automation Conference (DAC)*, 2013, pp. 115:1–115:7.

[15] T. Theocharides, M. K. Michael, M. Polycarpou, and A. Dingankar, "Hardware-enabled Dynamic Resource Allocation for Manycore Systems Using Bidding-based System Feedback," *EURASIP J. Embedded Syst.*, vol. 2010, pp. 3:1–3:21, 2010.

[16] N. Bansal and K. R. Pruhs, "Server Scheduling to Balance Priorities, Fairness, and Average Quality of Service," *SIAM J. Comput.*, vol. 39, no. 7, pp. 3311–3335, 2010.

[17] P. Li and B. Ravindran, "Fast, Best-Effort Real-Time Scheduling Algorithms," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1159–1175, 2004.

[18] S. Aldarmi and A. Burns, "Dynamic value-density for scheduling real-time systems," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 1999, pp. 270–277.

[19] A. K. Singh, P. Dziurzanski, and L. S. Indrusiak, "Market-inspired Dynamic Resource Allocation in Many-core High Performance Computing Systems," in *IEEE International Conference on High Performance Computing & Simulation (HPCS)*, 2015, pp. 413–420.

[20] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of Conference on Power Aware Computing and Systems (HotPower)*, 2008, pp. 10–10.

[21] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurr. Comput. : Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, 2012.

[22] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger, "Correlation-aware Virtual Machine Allocation for Energy-efficient Datacenters," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2013, pp. 1345–1350.

[23] Y. Gao, Y. Wang, S. Gupta, and M. Pedram, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, 2013, pp. 1–10.

[24] S. Eyerman and L. Eeckhout, "Fine-grained DVFS Using On-chip Regulators," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 1, pp. 1:1–1:24, 2011.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.