Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget

Osman Sarood, Akhil Langer, Abhishek Gupta, Laxmikant Kale
Department of Computer Science
University of Illinois Urbana-Champaign, Urbana, IL, USA
{sarood1, alanger, gupta59, kale}@illinois.edu

***Abstract*—Building future generation supercomputers while constraining their power consumption is one of the biggest challenges faced by the HPC community. For example, US Department of Energy has set a goal of 20 MW for an exascale (10^{18} flops) supercomputer. To realize this goal, a lot of research is being done to revolutionize hardware design to build power efficient computers and network interconnects. In this work, we propose a software-based online resource management system that leverages hardware facilitated capability to constrain the power consumption of each node in order to optimally allocate power and nodes to a job. Our scheme uses this hardware capability in conjunction with an adaptive runtime system that can dynamically change the resource configuration of a running job allowing our resource manager to re-optimize allocation decisions to running jobs as new jobs arrive, or a running job terminates.**

We also propose a performance modeling scheme that estimates the essential power characteristics of a job at any scale. The proposed online resource manager uses these performance characteristics for making scheduling and resource allocation decisions that maximize the job throughput of the supercomputer under a given power budget. We demonstrate the benefits of our approach by using a mix of jobs with different power-response characteristics. We show that with a power budget of 4.75 MW, we can obtain up to 5.2X improvement in job throughput when compared with the SLURM scheduling policy that is power-unaware. We corroborate our results with real experiments on a relatively small scale cluster, in which we obtain a 1.7X improvement.

I. Introduction

The US Department of Energy has identified four key areas that require significant research breakthroughs in order to achieve exascale computing over the next decade [1]. This paper explores a global power monitoring strategy for high performance computing (HPC) data centers and addresses one of those four key areas – machine operation under power envelope of 20MW. Our work focuses on a hardware-assisted software-based resource management scheme that intelligently allocates nodes and power to jobs with the aim of maximizing the job throughput, under a given power budget. Most of the current and past work in HPC is done from the perspective that number of nodes is the limiting factor which can freely draw any amount of power. As we move on to large machines, power consumption will overtake hardware as the limiting factor. Hence, HPC researchers might be forced to think power as the limiting factor. This paper proposes an HPC scheduling system that considers power to be the limiting resource and hence assumes that it is economical to add nodes to the supercomputer to efficiently utilize the power budgeted for the data center.

Computer subsystems such as CPU and memory have a vendor-specified Thermal Design Power (TDP) that corresponds to the maximal power draw by the subsystem. Currently, maximum power consumption of an HPC data center is determined by the sum of the TDP of its subsystems. This is wasteful, as these subsystems seldom run at their TDP limit. Nonetheless, near TDP amount of power has to be set aside for the subsystems so that the circuit breakers do not trip in that rare case when the power consumption reaches TDP. Recent advances in processor and memory hardware designs have made it possible for the user to control the power consumption of the CPU and memory through software, e.g., the power consumption of Intel's Sandy Bridge family of processors can be user-controlled through the Running Average Power Limit (RAPL) library [2]. Some other examples of such architectures are IBM Power6, Power7, and AMD Bulldozer. This ability to constrain the maximum power consumption of the subsystems, below the vendor-specified TDP value, allows us to add more machines while ensuring that the total power consumption of the data center does not exceed its power budget. Such a system is called an *overprovisioned* system [3].

Earlier work [3], [4] shows that an increase in the power allowed to the processor (and/or memory) does not yield a proportional increase in the application's performance. As a result, for a given power budget, it can be better to run an application on larger number of nodes with each node capped at lower power than fewer nodes each running at its TDP. The optimal resource configuration for an application can be determined by profiling an application's performance for varying number of nodes, CPU power and memory power and then selecting the best performing configuration for the given power budget [4]. In this work, we address the data center scenario in which an additional decision has to be made: how to distribute available nodes and power amongst the queued jobs. The challenge is to obtain a distribution that is globally optimal and not individually for the jobs (which was the focus of the earlier work [4]). Additionally, new job requests arrive with time and currently running jobs terminate, which requires re-optimization of scheduling and resource allocation decisions. With this context, we believe the major contributions of this paper are:

- An online resource manager, PARM, that uses overprovisioning, power capping and job malleability along with the power-response characteristics of each job for scheduling and resource allocation decisions that significantly improve the job throughput of the data center (§ IV).

- A power aware strong scaling (PASS) performance model that estimates an applications performance for a given number of nodes and CPU power cap (§ V). We demonstrate the use of our model by estimating characteristics of five applications having different power-response characteristics (§ VI-C).
- An evaluation of our online resource manager on a 38 node cluster with two different job data sets. A speedup of 1.7 was obtained when compared with SLURM (§ VI).
- An extensive simulated evaluation of our scheduling policy for larger machines and its comparison with the SLURM baseline scheduling policy. We achieve up to 5.2X speedup operating under a power budget of 4.75 MW (§ VII).

II. Related work

Performance modeling using Dynamic Voltage and Frequency Scaling (DVFS) has been extensively studied before [5]–[8]. These models estimate execution time based on the CPU frequency. These models cannot be used directly in the context of CPU power capping. This is because applications running under the same CPU power cap can have different CPU frequencies because of the difference in their memory/CPU characteristics. Based on the on-chip activity of the application, the frequency will be adjusted in order to ensure the CPU power cap. The power aware strong scaling model proposed in this paper differs from the previous work as it estimates execution time for a given CPU power cap (that includes power consumption of cores, caches, and memory controller present on the chip).

Power and energy profiling of parallel scientific workload [9]–[11] and modeling of their energy efficiency [12] has been extensively researched before. Their has been a lot of work on reducing the energy consumption by applications while minimizing the degradation in performance. For instance, Porterfield et al in [13] conserve energy consumption by using an adaptive runtime system to automatically throttle the number of concurrently used threads based on power and energy consumption gathered by using the RAPL interface. Mammela [14] et al have proposed a scheduling scheme to reduce energy consumption of an entire data center using DVFS. Other work on energy optimization can be found in [15]–[18]. Our work deals with the current scenario in which performance has to be optimized with power as a hard constraint as compared to optimizing energy efficiency.

Patki et al [3] proposed the idea of overprovisioning the compute nodes in power-constrained HPC data centers. Application is profiled at various node levels and CPU power caps. For a given power budget, the best operating configuration is then selected from the sampled configurations. Sarood et.al. [4] with average system load of 7. included memory power capping and proposed a scheme to obtain exhaustive profiling of an application, thus improving the performance by selecting the optimal configuration for a given power budget. In this work, we propose an online scheduler for an overprovisioned data center, that optimizes the distribution of power and nodes to multiple jobs selected by the scheduler for simultaneous execution. To the best of our knowledge, PARM is the first resource manager that uses CPU power capping and job malleability to show significant improvements in the throughput of the data center.

III. Data Center and Job Capabilities

In this section, we describe some of the capabilities or features which, according to our understanding, ought to be present in future HPC data centers. In the following sections, we highlight the role these capabilities play for a scheduler, whose goal is to increase the throughput of a data center while ensuring fairness. **Power capping**: This feature allows the scheduler to constrain the individual power draw of each node. Intel's Sandy Bridge processor family supports on-board power measurement and capping through the RAPL interface [2]. RAPL is implemented using a series of Machine Specific Registers (MSRs) which can be accessed to read power usage for each power plane. RAPL supports power capping Package and DRAM power planes by writing into the relevant MSRs. Here, 'Package' corresponds to the processor chip that hosts processing cores, caches and memory controller. In this paper, we use package power interchangeably with CPU power, for ease of understanding. RAPL can cap power at a granularity of milliseconds which is adequate given that the capacitance on the motherboard and/or power supply smoothes out the power draw at a granularity of seconds. **Overprovisioning**: By using RAPL to cap the CPU (same as package) power below TDP value, it is possible to add more nodes to the data center while staying within the power budget. An overprovisioned system is thus defined as a system that has more nodes than a conventional system operating under the same power budget. Due to the additional nodes, such a system can not enable all of its nodes to function at their maximum TDP power levels simultaneously. **Moldable jobs**: In these jobs, the user specifies the range of nodes (the minimum and the maximum number of nodes) on which the job can run. Typically, the range of the nodes in which the job can run is dictated by its memory usage and strong scaling characteristics. The job scheduler decides the number of nodes within the specified range to be allocated to the job. Once decided, the number of nodes cannot be changed during job execution [19]. **Malleable jobs**: Such jobs can shrink to a smaller number of nodes or expand to a larger number of nodes upon instruction from an external command with in a specified range. The number of allocated nodes to a malleable job can be changed at any time during the execution of the job. To enable malleable jobs, two components are critical – *a smart job scheduler*, which decides when and which jobs to shrink or expand, and a *parallel runtime system* which provides dynamic shrink and expand capability to the job. We rely on existing runtime support for malleable jobs in Charm++ [20]–[22]. In Charm++, malleability is achieved by dynamically redistributing compute objects to processors at runtime. Applications built on top of such an adaptive system have been shown to shrink and expand with small costs [23]. Charm++ researchers are currently working on further improving the support for malleable jobs. Prior research has also shown how MPI applications can be made malleable. Some notable works are dynamic CPUSET's mapping and dynamic MPI [24], dynamic malleability of iterative MPI applications using PCM (Process Checkpoint and Migration) library [25], and migratable threads-based overdecomposition and disk-based checkpoint-restart in Adaptive MPI [26].

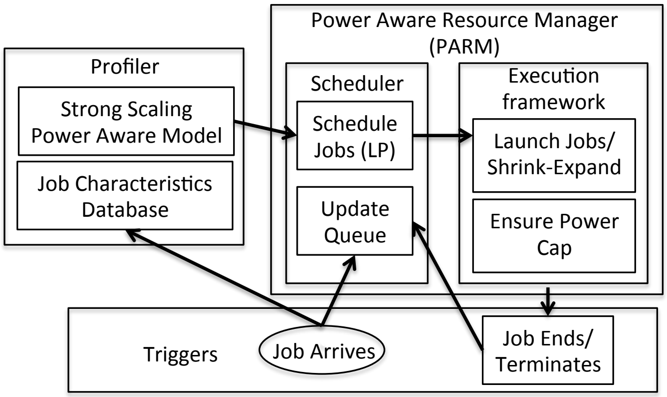


Fig. 1: A high level overview of PARM

IV. Power Aware Resource Manager

Figure 1 shows the block diagram of our online Power Aware Resource Manager, or PARM. It has two major modules: the scheduler and the execution framework. The scheduler is responsible for identifying which jobs should be scheduled and exactly which resources should be devoted to each job. We refer to the resource allocation for each job by the *resource combination* tuple, (n, p), where n is the number of nodes and p is the CPU power cap for each of the n nodes. The scheduling decision is made based on the Integer Linear Program (ILP), and the job profiles generated by our strong scaling power aware model described in § V. The scheduler's decisions are fed as input to the execution framework which implements/enforces them by launching new jobs, shrinking/expanding running jobs, and/or setting the power caps on the nodes.

The scheduler is triggered whenever a new job arrives or when a running job ends or abruptly terminates due to an error or any other reason ('Triggers' box in Figure 1). At each *trigger*, the scheduler tries to re-optimize resource allocation to the set of pending as well as currently running jobs with the objective of maximizing overall throughput. Our scheduler uses both CPU power capping and moldability/malleability features for throughput maximization. We formulate this resource optimization problem as an Integer Linear Program (ILP). The relevant terminology is described in Table I. Our scheduling scheme can be summarized as:
Input: A set of jobs that are currently executing or are ready to be executed ($\mathcal{J}$) with their expected execution time corresponding to a set of resource combinations (n, p), where $n \in N_j$ and $p \in P_j$.
Objective: Maximize data center throughput.
Output: Allocation of resources to jobs at each trigger event, i.e., identifying the jobs that should be executed along with their resource combination *(n,p)*.

A. Integer Linear Program Formulation

We make the following assumptions and simplifications in the formulation:

- All nodes allocated to a given job operate at the same power.
- We do not include cooling power of the data center in our calculations.
- Job characteristics do not change significantly during the course of its execution. By relaxing this assumption we can benefit from the different phases in an application. However, that is out of the scope of this study.
- The network power consumption stays constant. It is a reasonable assumption since network power does not fluctuate

Objective Function

$$\sum_{j \in \mathcal{J}} \sum_{n \in N_j} \sum_{p \in P_j} w_j * s_{j,n,p} * x_{j,n,p} \tag{1}$$

Select One Resource Combination Per Job

$$\sum_{n \in N_j} \sum_{p \in P_j} x_{j,n,p} \leq 1 \qquad \forall j \in I \tag{2}$$

$$\sum_{n \in N_j} \sum_{p \in P_j} x_{j,n,p} = 1 \qquad \forall j \in \mathcal{I} \tag{3}$$

Bounding total nodes

$$\sum_{j \in \mathcal{J}} \sum_{p \in P_j} \sum_{n \in N_j} n x_{j,n,p} \leq \mathbf{N} \tag{4}$$

Bounding power consumption

$$\sum_{j \in \mathcal{J}} \sum_{n \in N_j} \sum_{p \in P_j} (n * (p + W_{base})) x_{j,n,p} \leq W_{max} \tag{5}$$

Disable Malleability (Optional)

$$\sum_{n \in N_j} \sum_{p \in P_j} n x_{j,n,p} = n_j \qquad \forall j \in \mathcal{I} \tag{6}$$

Fig. 2: Integer Linear Program formulation of PARM scheduler

TABLE I: Integer Linear Program Terminology

Symbol	Description
$\mathbf{N}$	total number of nodes in the data center
J	set of all jobs
$\mathcal{I}$	set of jobs that are currently running
I	set of jobs in the pending queue
$\mathcal{J}$	set of jobs which have already arrived and have not yet been completed i.e they are either pending or currently running, $\mathcal{J} = \mathcal{I} \cup I$
N_j	set of node counts on which job j can be run
P_j	set of power levels at which job j should be run or in other words, the power levels at which job j's performance is known
n_j	number of nodes at which job j is currently running
w_j	weighing factor to set job priorities
α	a constant in w_j used to tradeoff job fairness/priority vs data center throughput
$x_{j,n,p}$	binary variable, 1 if job j should run on n nodes at power p, otherwise 0
t_{now}	current time
t_j^a	arrival time of job j
W_{base}	base machine power that includes everything other than CPU and memory
$t_{j,n,p}$	execution time for job j running on n nodes with power cap of p
$s_{j,n,p}$	strong scaling power aware speedup of application j running on n nodes with power cap of p

much for most interconnect technologies.

- Expected wall clock time represents a good estimate of the actual execution time that the scheduler uses for decision making.
- W_{base}, that includes power for all the components of a node other than the CPU and memory subsystems, is assumed to be constant.
- A job once selected for execution is not stopped until its completion, although the resources assigned to it can change during its execution.
- All jobs are from a single user (or have the same priority). This is assumed just to keep the focus of the paper on other issues. This assumption can be very easily relaxed by setting w_j proportional to the user/job-queue priority.

Scheduling problems are framed as ILPs and ILPs are NP-hard problems. Maximizing throughput in the objective function requires introducing variables for the start and end time of jobs. These variables make the ILP computationally very intensive and thus impractical for online scheduling in many cases. Therefore in the objective function, instead of maximizing the overall throughput of all the jobs currently in queue, we propose a greedy objective function that maximizes the sum of the power-aware speedup (described later) of the jobs selected for immediate execution. This objective function improves the job throughput while keeping the ILP optimization computationally tractable for online scheduling.

We define the strong scaling power aware speedup of a job j as follows:

$$s_{j,n,p} = \frac{t_{j,min(N_j),min(P_j)}}{t_{j,n,p}} \tag{7}$$

where $s_{j,n,p}$ is the speedup of job j executing using resource combination (n, p) with respect to its execution with resource combination $(min(N_j), min(P_j))$. Objective function (Eq. 1) of the ILP maximizes the sum of the power aware speedups of the jobs selected for execution at every trigger event. This leads to improvement in FLOPS/Watt (or power efficiency, as we define it). Improved power efficiency implies better job throughput (results discussed in § VI,§ VII). Oblivious maximization of power efficiency may lead to starvation for jobs with low strong scaling power aware speedup. Therefore, to ensure fairness, we introduced a weighing factor (w_j) in the objective function, which is defined as follows:

$$w_j = (t^{rem}_{j,min(N_j),min(P_j)} + (t_{now} - t^a_j))^\alpha \tag{8}$$

w_j artificially boosts the strong scaling power aware speedup of a job by multiplying it by the job's completion time, where completion time is the sum of the time elapsed since job's arrival and the job's remaining execution time with resource combination $(min(N_j), min(P_j))$ i.e. $(t^{rem}_{j,min(N_j),min(P_j)})$. The percentage of a running job completed between two successive triggers is determined by the ratio of the time interval between the two triggers and the total time required to complete the job using its current resource combination. Percentage of the job that has been completed so far can then be used to compute $t^{rem}_{j,min(N_j),min(P_j)}$. The constant α ($\alpha \geq 0$) in Eq. 8 determines the priority given to job fairness against its strong scaling power aware speedup i.e. a smaller value of α favors job throughput maximization while a larger value favors job fairness. We now explain the constraints of the ILP formulation (Figure 2):

TABLE II: Different versions of PARM

Acronym	Description
noMM	Jobs are neither Moldable nor Malleable
noSE	Jobs are moldable but not malleable
wSE	Jobs are both moldable and malleable

- Select one resource combination per job (Eq. 2,3): $x_{j,n,p}$ is a binary variable indicating if job j should run using resource combination (n, p). This constraint ensures that at most one of the variables $x_{j,n,p}$ is set to 1 for any job j. The jobs which are already running (set $\mathcal{I}$) continue to run although they can be assigned a different resource combination (Eq. 3). The jobs in the pending queue (I), for which at least one of the variables $x_{j,n,p}$ is equal to 1 (Eq. 2), are selected for execution and moved to the set of jobs currently running ($\mathcal{I}$).
- Bounding total nodes (Eq. 4): This constraint ensures that the number of active nodes do not exceed the maximum number of nodes available in the overprovisioned data center.
- Bounding power consumption (Eq. 5): This constraint ensures that power consumption of all the nodes does not exceed the power budget of the data center.
- Disable Malleability (Eq. 6): To quantify the benefits of malleable jobs, we consider two versions of our scheduler. The first version supports only moldable jobs and is called noSE (i.e. no Shrink/Expand). The second version allows both moldable and malleable jobs and is called as wSE (i.e. with Shrink/Expand). Malleability can be disabled by using Eq. 6. This constraint ensures that number of nodes assigned to running jobs does not change during the optimization process. However, it allows changing the power allocated to running jobs. In real-world situations, the jobs submitted to a data center will be a mixture of malleable and non-malleable jobs. The scheduler can apply Eq. 6 to disable malleability for non-malleable jobs. In addition to the noSE and wSE, we also measure the performance of noMM (no Moldability and Malleability) version of PARM in which the jobs are neither moldable nor malleable. In this version, besides job selection, the only degree of freedom available to PARM is the CPU power allocated to the nodes of the selected jobs. The three versions of the PARM are summarized in Table II for ease of reference.

V. Power Aware Strong Scaling Performance Model

PARM's optimal resource allocation decisions depend on the availability of jobs performance data. Performance data corresponding to a large number of resource combinations (n, p) can be crucial to the quality of solution PARM provides. Since exhaustive profiling can be impractical for large number of resource combinations, we need a model to predict job performance. One of the significant contributions of our work is the proposed performance model that can predict an application's performance for any given resource combination (n, p). We call it a Power Aware Strong Scaling performance

TABLE III: Power Aware Strong Scaling Model Terminology

Symbol	Description
A	Average parallelism in the application
σ	fraction of the duration when application parallelism is not A, parallelism is $2A-1$ for $\frac{\sigma}{2}$ fraction and 1 for $\frac{\sigma}{2}$ fraction of the duration
T_1	Application execution time on 1 node
f	CPU Frequency
f_h	Threshold frequency beyond which application execution time does not reduce
f_l/f_{min}	Minimum CPU frequency supported by vendor
f_{max}	Maximum CPU frequency supported by vendor
T_l	Execution time at CPU frequency f_l
T_h	Execution time at CPU frequency f_h
W_{cpu}	on-chip workload in terms of CPU cycles
T_{mem}	Time for off-chip work in the application that is unaffected by CPU frequency

model or PASS model. The model parameters are specific to the application and the input dataset with which the application will be executed. Applying mathematical regression to application's profile data for different resource combinations enables PASS to estimate important power characteristics. PASS model extends Downey's [27] strong scaling model by making it power aware. Table III gives the terminology used in this section.

A. Strong Scaling Model

An application can be characterized by an average parallelism of A. The application's parallelism remains equal to A, except for some fraction σ of the duration. Available parallelism is $2A-1$ for $\frac{\sigma}{2}$ fraction of the duration and just 1 for the remaining $\frac{\sigma}{2}$ fraction of the duration. We adjust Downey's [27] model to satisfy the boundary conditions - $t(1) = T_1$, and $t(n) = \frac{T_1}{A}$ for $n \geq A$, where $t(n)$ is the application time on n nodes, and T_1 is the application time on a single node. According to Downey's model, the execution time, $t(n)$, of an application executing on n nodes can be modeled as:

$$t(n) = \begin{cases} \dfrac{T_1 - \frac{T_1\sigma}{2A}}{n} + \dfrac{T_1\sigma}{2A}, & 1 \leq n \leq A \qquad (9) \\ \dfrac{\sigma(T_1 - \frac{T_1}{2A})}{n} + \dfrac{T_1}{A} - \dfrac{T_1\sigma}{2A} & A < n \leq 2A-1 \qquad (10) \\ \dfrac{T_1}{A}, & n > 2A-1 \qquad (11) \end{cases}$$

The first equation in this group represents the range of n where the application is most scalable i.e. when the number of nodes is less than A. The application's scalability declines significantly once n becomes larger than A because of lack of parallelism for most of the duration. Finally, for $n \geq 2A$, the execution time $t(n)$ equals T_1/A and does not decrease further. Given application characteristics σ, A, and T_1, this model can be used to estimate execution time for any number of nodes n.

B. Adding Power Awareness to Strong Scaling Model

The effect of changing frequency on the execution time varies from application to application [28] . In this section, we model execution time as a function of CPU frequency. Since, CPU frequency can be expressed as a function of CPU power, we can finally express execution time as a function to CPU power.

1) Execution Time as a Function of Frequency: Existing work [4], [28] indicates that increase in CPU frequency beyond a certain threshold frequency (let us call it f_h) does not reduce the execution time. The value of f_h depends on the memory bandwidth being used by the application. For $f < f_h$, execution time depends on the CPU-bounded and memory (off-chip) bounded work of the application and can thus be modeled as [5]–[7], [29]:

$$t(f) = \begin{cases} \dfrac{W_{cpu}}{f} + T_{mem}, & \text{for } f < f_h \qquad (12) \\ T_h, & \text{for } f \geq f_h \qquad (13) \end{cases}$$

where, W_{cpu} and T_{mem} are defined in Table III, and T_h is the execution time at frequency f_h. Let T_l be the execution time at frequency f_l where f_l is the minimum frequency at which the CPU can operate. Parameter β characterizes the frequency-sensitivity of an application and can be expressed as:

$$\beta = \frac{T_l - T_h}{T_l} \qquad (14)$$

Range of β depends on the frequency range supported by the CPU vendor. Given the frequency range of (f_l, f_{max}), $\beta \leq 1 - \frac{f_l}{f_{max}}$. Typically, CPU-bound applications have higher values for β whereas memory-intensive applications have smaller β values.

Using Eq. 14 and applying boundary conditions, $t(f_l) = T_l$ and $t(f_h) = T_h$, to Eq. 12, we get:

$$W_{cpu} = \frac{T_h \beta f_l f_h}{(1-\beta)(f_h - f_l)} \qquad (15)$$

$$T_{mem} = T_h - \frac{T_h \beta f_l}{(1-\beta)(f_h - f_l)} \qquad (16)$$

2) Frequency as a Function of CPU Power: Although Intel has not released complete details of how the CPU power consumption is ensured to be below the user specified CPU power cap, it has been hinted that it is achieved using a combination of DVFS and CPU throttling [2], [30].

Let p_l denote the CPU power corresponding to f_l, where f_l is the minimum frequency the CPU can operate at using DVFS. To cap power below p_l ($p < p_l$), other architectural-level mechanisms such as CPU throttling are used. We have empirically observed that for $p < p_l$, the application performance degrades significantly even for very small savings in power. Therefore, we restrict our study to power caps greater than p_l. The value of p_l can be easily determined by setting the CPU frequency at f_l. CPU or the package power includes the power consumption by its various components such as cores, caches, memory controller, etc. The value of p_l varies depending on an application's usage of these components.

In a CPU-bound application, a processor might be able to cap power to lower values using DVFS, since only the cores are consuming power. In contrast, for a memory intensive application, p_l might be higher, since the caches and memory controller are also consuming significant power in addition to the cores.

The major part of the dynamic CPU power consumption can be attributed to the cores, on-chip caches and memory controller. Power consumption of the core, p_{core}, is often modeled as $p_{core} = Cf^3 + Df$, where C and D are some constants [31]. Power consumption due to cache and memory accesses is modeled as, $\sum_{i=1}^{3} g_i L_i + g_m M$, where, L_i is accesses per second to level i cache, g_i is the cost of a level i cache access, M is the number of memory accesses per second, g_m is the cost per memory access. The total CPU power can then be expressed as [32]:

$$p = p_{core} + \sum_{i=1}^{3} g_i L_i + g_m M + p_{base} \tag{17}$$

where, p_{base} is the base/static package power consumption. Since number of cache and memory accesses is proportional to the CPU frequency, Eq. 17 can be written as:

$$p = F(f) = af^3 + bf + c \tag{18}$$

where a, b, and c are constants. bf corresponds to the cores' leakage power and power consumption of caches and memory controller. The term af^3 represents the dynamic power of the cores, whereas, $c = p_{base}$ represents the base CPU power. The constants a and b are application dependent since the cache and memory behavior can be different across applications. Eq. 18 can be rewritten as a depressed cubic equation and solved using Fermat's Last Theorem to get F^{-1}:

$$\begin{aligned} f = F^{-1}(p) \quad &= \quad \sqrt[3]{\frac{p-c}{2a} + \sqrt{\frac{(p-c)^2}{4a^2} + \frac{b^3}{27a^3}}} \\ &+ \quad \sqrt[3]{\frac{c-p}{2a} + \sqrt{\frac{(p-c)^2}{4a^2} + \frac{b^3}{27a^3}}} \end{aligned} \tag{19}$$

3) Execution Time as Function of CPU power & Number of Nodes: To express t in terms of p, we use Eq. 19 to replace f, f_l, and f_h in Eqs. 12, 15, 16. To obtain the PASS model, that estimates execution time as a function of n and p, we combine our power aware model with the strong scaling model described in § V-A, by replacing T_h in Eqs. 12, 13, 15, 16 with $t(n)$ from Eqs. 9, 10, 11.

VI. Experimental Results

In this section, we first describe our experimental setup that includes applications, testbed, and job datasets. Next, we obtain the application characteristics using the PASS performance model and finally compare the performance of different versions of PARM with SLURM. PARM can be used in conjunction with most parallel programming models. While programming models such as CHARM++ can benefit by using wSE scheme that uses job malleability, other models like MPI, can use the noSE scheme to benefit from power awareness and job moldability. Usage of PARM is not restricted to data centers with focus on running large number of applications simultaneously. It can even be used in data centers where performance of running very small number (or just 1) of large-scale applications is critical. For example, while running just 1 large job, PARM optimizer will determine the optimal number of nodes and the cpu power cap of the nodes, on which the job should be executed for optimal performance.

A. Applications

We used five applications, namely, Wave2D, Jacobi2D, LeanMD, Lulesh [33], and Adaptive Mesh Refinement or AMR [34], [35]. These applications have different CPU and memory usage:

- Wave2D and Jacobi2D are 5-point stencil applications that are memory-bound. Wave2D has higher FLOPS than Jacobi2D.
- LeanMD is a computationally intensive molecular dynamics application.
- CPU and memory usage of Lulesh and AMR lies in between the stencil applications and LeanMD.

B. Testbed

We conducted our experiments on a 38-node Dell PowerEdge R620 cluster (which we call the Power Cluster). Each node contains an Intel Xeon E5-2620 Sandy Bridge with 6 physical cores at 2GHz, 2-way SMT with 16 GB of RAM. These machines support on-board power measurement and capping through the RAPL interface [36]. The CPU power for our testbed can be capped in the range $[25-95]W$, while the capping range for memory power is $[8-35]W$.

C. Obtaining Model Parameters of Applications

Application characteristics depend on the input type, e.g., grid size. We fix the respective input types for each application. Each application needs to be profiled for some (n, p) combinations to obtain data for curve fitting. A single time step (iteration) of an application is sufficient to get the performance for a given resource combination. For applications having time steps (iteration) in order of milliseconds, the cost of profiling several resource combinations is negligible compared to the overall execution time of the application. Each time step (iteration) will include the different phases of an application such as IO, communication, solvers, etc. and therefore the overall characteristics of the job can be captured in a step/iteration. This approach works best for iterative applications and other applications whose characteristics do not change significantly over time, which is true for majority of the scientific applications.

We use linear and non-linear regression tools provided by MATLAB to determine the application parameters by fitting our performance model proposed in § V to the sampled performance data obtained by running the parallel applications on 20 nodes. The obtained parameter values for all the applications are listed in Table IV and are discussed here:

- The parameter c (CPU base power) lies in the range $[13-14]$W for all applications
- p_l was 30W for LeanMD and $32W$ for rest of the applications. For LeanMD, it is possible to cap the CPU power to a lower value just by decreasing the frequency using DVFS.

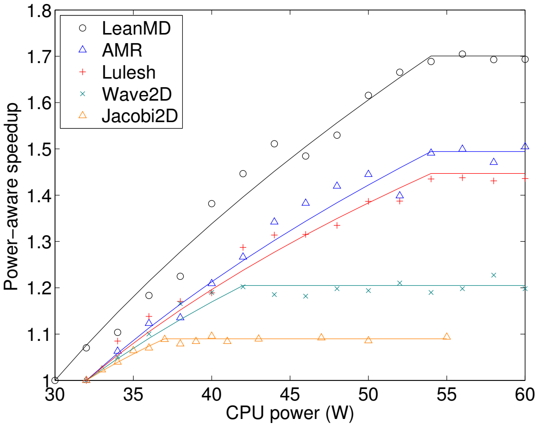


Fig. 3: Modeled (lines) and observed (markers) power aware speedups of the applications at 20 nodes

TABLE IV: Obtained model parameters

Application	**a**	**b**	p_l	p_h	β
LeanMD	1.65	7.74	30	52	0.40
Wave2D	3.00	10.23	32	40	0.16
Lulesh	2.63	8.36	32	54	0.30
AMR	2.45	6.57	32	54	0.33
Jacobi2D	1.54	10.13	32	37	0.08

This is because LeanMD is a computationally intensive application and therefore most of the power is consumed by the cores rather than caches and memory controller. On the contrary, for other applications, CPU throttling kicks in at a higher power level because of their higher cache/memory usage.

- value of p_h lies in the range of $[37 - 54]W$ for the applications under consideration.
- value of β lies in the range $[0.08 - 0.40]$. Higher value of β means higher sensitivity to CPU power.
- Wave2D and Jacobi2D have the largest memory footprint that results in high CPU-cache-memory traffic. Therefore the value of b is high for these two applications.

Figure 3 shows the modeled (lines) as well as the observed (markers) power-aware speedups for all applications with varying CPU power cap at 20 nodes. Power-aware speedup is calculated with respect to the execution time at $p = p_l$ and the same number of nodes. LeanMD has the highest power-aware speedup whereas Jacobi2D has the lowest.

D. Power Budget

We assume a power budget of 3300W to carry out experiments using our Power Cluster. Although the vendor-specified TDP of CPU and memory of the Dell nodes was 95W and 35W, respectively, the actual power consumption of CPU and memory never went beyond 60W and 18W while running any of the applications. Therefore, instead of the vendor-specified TDP, we consider 60W and 18W as the maximum CPU and memory power consumption and use them to calculate the number of nodes that can be installed in a traditional data center. The maximum power consumption of a node, thus, adds up to $60W + 18W + 38W = 116W$, where $38W$ is the base power of a node. Therefore, the total number of nodes that can be installed in a traditional data center with a power budget of 3300W will be $\lfloor \frac{3300}{116} \rfloor = 28$ nodes. By capping the CPU power below 60W, the overprovisioned data center will be able to power more than 28 nodes.

E. Job Datasets

We constructed two job datasets by choosing a mix of applications from the set described in § VI-A. All these applications are written using the Charm++ parallel programming model and hence support job malleability. Application's power-response characteristics can influence the benefits of PARM. Therefore, in order to better characterize the benefits of PARM, these two job datasets were constructed such that they have very different average values of β. We name these datasets as SetL and SetH, with average β value of 0.1 and 0.27, respectively. For instance, SetH has 3 LeanMD, 3 Wave2D, 2 Lulesh, 1 Jacobi, and 1 AMR job, that gives us an average β value of 0.27. A mix of short, medium and long jobs were constructed by randomly generating wall clock times with a mean value of 1 hour. Similarly, the job arrival times were generated randomly. Each dataset spans over 5 hours of cluster time and approximately 20 scheduling decisions were taken (a scheduling decision is taken whenever a new job arrives or a running job terminates). The minimum and the maximum number of nodes on which a job can run was determined by the job's memory requirements. We used 8 node levels (i.e. $|N_j|= 8$) that are uniformly distributed between the minimum and maximum number of nodes on which the job can run. The memory power is capped at the fixed value of 18W whereas we used 6 CPU power levels - $[30, 32, 34, 39, 45, 55]$W.

F. Performance Metric

We compare our scheduler with SLURM [37]: an open-source resource manager that allocates compute nodes to jobs and provides a framework for starting, executing and monitoring jobs on a set of nodes. SLURM provides resource management on many of the most powerful supercomputers of the world including Tianhe-1A, Tera 100, Dawn, and Stampede. We deployed both PARM and SLURM on the testbed. For comparison purpose, we use SLURM's scheme in which the user specifies the exact number of nodes requested for the job and SLURM uses FIFO + backfilling for making scheduling decisions. We call this as the SLURM baseline scheme or just the baseline scheme. The number of nodes requested for a job submitted to SLURM is the minimum number of nodes on which PARM can run that job.

We use response time and completion time as the metric for comparing PARM and SLURM. A job's response time, t_{res}, is the time interval between its arrival and the beginning of its execution. Execution time, t_{exe}, is the time from start to finish of a job's execution. Completion time, t_{comp}, is the time between job's arrival and the time it finished execution, i.e., $t_{comp} = t_{res} + t_{exe}$. Job throughput is the inverse of the average completion time of jobs. In this study, we emphasize on completion time as the performance comparison metric, even though typically response time is the preferred metric. This is because unlike conventional data centers, where resources allocated to a job and hence the jobs execution time are fixed, our scheduler dynamically changes job configuration during execution which can vary job execution time significantly. Hence, response time is not a very appropriate metric for comparison in this study. Completion time includes both the response time and the execution time and is therefore the preferred metric of comparison.

G. Results

Figure 4(a) shows the average completion times of the two datasets with SLURM and noMM, noSE, and wSE versions of PARM. In noMM experiments, the number of nodes allocated to a job were the minimum number of nodes on which the noSE and wSE versions can run the same job. The maximum number of nodes on which noSE and wSE can run the job were same as the number of nodes on which SLURM runs the same job. The completion times for noMM, wSE and noSE include all overhead costs including the ILP optimization time and the costs of constriction and expansion of jobs. All versions of PARM significantly reduce the average completion time for both the data sets compared to SLURM (Figure 4(a)). This improvement can mainly be attributed to the reduced average response times shown in Figure 4(b). Our scheduler intelligently selects the best power levels for each job which allows it to add more nodes to benefit from strong scaling and/or scheduling more jobs simultaneously. noMM version of PARM significantly reduces the completion times by intelligently selecting jobs and allocating power to them. Allowing job moldability and malleability in noSE and wSE, respectively, further reduces the completion times. For example, there is an improvement of 7.5% and 13.9% in average completion time of SetL with noSE and wSE, respectively over the noMM scheme. A speedup of 1.7X is achieved with wSE scheme over the baseline scheme. Ability to shrink and expand the jobs for wSE version of PARM, gives additional flexibility to the ILP to re-optimize the allocation of nodes to the running and pending jobs. noSE reduces the solution space of ILP (compared to wSE) by not allowing running jobs to change the number of nodes allocated to them during their execution. The noMM version reduces the search space even further by fixing the number of nodes allocated to a job both at the start time and during its execution. The wSE version further benefits from the fact that it can expand the running jobs to run on the unutilized machines. e.g. when there are not enough jobs to utilize all the nodes. These factors reduce both the average completion and the average response time in wSE (Figure 4(a), 4(b)). As shown by Figure 4(c), wSE scheme utilizes an average of 36 nodes during the entire dataset execution as compared to an average of 33 nodes used in the case of noSE for SetL.

A smaller value of β means that the effect of decreasing the CPU power on application performance is small. When β is small, the scheduler will prefer to allocate less CPU power and use more nodes. When β is large, the benefits of adding more nodes at the cost of decreasing the CPU power are smaller. The flexibility to increase the number of nodes gives PARM higher benefit over SLURM when β is small as compared to the case when β is large. This is corroborated with the observation (Figure 4) that the benefits of using PARM as compared to SLURM are much higher with dataset SetL ($\beta = 0.1$) as compared to dataset SetH ($\beta = 0.27$). PARM's intelligent allocation of power can significantly improve completion and response times. These can be further improved by using job moldability and malleability features.

VII. Large Scale Projections

After experimentally showing the benefits of PARM on a real cluster, we now analyze its benefit on very large machines. Since it was practically infeasible for us to do actual job scheduling on very large machine, we use the SLURM simulator [38] which is a wrapper around SLURM. This simulator gives us information about SLURM's scheduling decisions without actually executing the jobs. To make analysis of PARM more reliable, we develop a model to estimate the cost of shrinking and expanding jobs. We then give the experimental setup and present a comparison of PARM scheduling with baseline scheduling policy. Since noMM version of PARM was inferior to both wSE and noSE, we concentrate on wSE and noSE schemes in this section.

A. Modeling Cost of Shrinking and Expanding Jobs

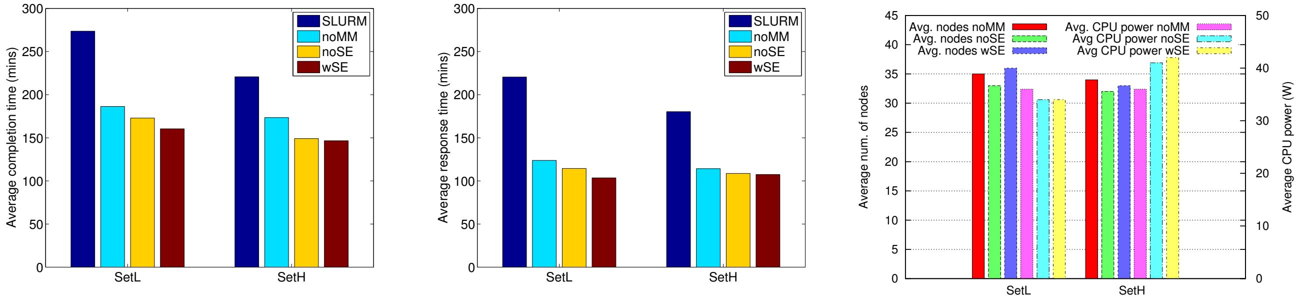
Constriction and expansion of jobs has an overhead associated with it. These overheads come from data communication done to balance the load across the new set of processors assigned to the job and from the boot time of nodes.

For demonstrating our system using real experiments (§ VI), we used the existing malleability support in Charm++ [23]. However, the approach in [23] is practical only for small clusters as it starts processes on as many nodes as the job can run on. Inter-job interference and security concerns make that approach impractical for large-clusters, where many jobs run simultaneously. Charm++ researchers have recently proposed a new approach which eliminates the need of spawning processes on all nodes and does not leave any residual processes after shrink. Hence, for more practical and accurate large-scale projections, we model an approach which would require dynamic process spawning when expanding. Hence, we consider boot times in our model.

A scheduler typically makes two decisions: 1) how many nodes to assign to each job, and 2) which nodes to assign to each job. We address the first decision in this paper and defer the second for future work. Let us say that job j with a total memory of m_j MB, has to expand from n_f nodes to n_t nodes. For simplification of analysis, we assume that each job is initially allocated a cuboid of nodes (with dimensions- $\sqrt[3]{n_f} \times \sqrt[3]{n_f} \times \sqrt[3]{n_f}$) interconnected through a 3D torus. After the expand operation, size of the cuboid becomes $\sqrt[3]{n_f} \times \sqrt[3]{n_f} \times \frac{n_t}{\sqrt[\frac{2}{3}]{n_f}}$. For load balance, the data in memory (m_j MB) will be distributed equally amongst the n_t nodes. Hence, the communication cost for the data transfer can be expressed as (secs):

$$t_c = \frac{(\frac{m_j}{n_f} - \frac{m_j}{n_t}) * n_f}{2 * b * n_f^{\frac{2}{3}}} \tag{20}$$

where b is the per link bandwidth in MB/sec. The numerator in Eq. 20 represents the total data to be transferred whereas the denominator represents the bisection bandwidth of the cuboid. Similarly, the cost of shrinking a job is determined by computing the cost of distributing the data of $n_f - n_t$ nodes equally across the final n_t nodes.



(a) Average completion time

(b) Average response time

(c) Average nodes and average power per node

Fig. 4: Comparing performance of SLURM with noMM, noSE, and wSE versions of PARM.

Boot times can be significant for some supercomputers. Since many supercomputers in Top500 [39] belong to the Blue Gene family, we include their boot time when evaluating our scheme. We adopt the following simple linear model to calculate the boot time (t_b) for expand operation based on Intrepid boot time data [40]:

$$t_b(\text{in seconds}) = (n_t - n_f) * 0.01904 + 72.73 \qquad (21)$$

In an expand operation, communication phase can start only after additional nodes become available. These additional nodes might have to be booted. Therefore the total cost of a shrink or expand operation is the sum of boot time and data transfer time, i.e., $t_{se} = t_c + t_b$. A job set for expansion might receive additional nodes from a job undergoing constriction in the same scheduling decision. Therefore, an expanding job has to wait until the shrinking job has released the additional resources. To simplify this analysis, we determine the maximum t_{se} from amongst the shrinking/expanding jobs (t_{se}^{max}) and add $2t_{se}^{max}$ to the execution times of all the jobs shrinking or expanding during the current scheduling decision. To control the frequency of constriction or expansion of a job, and consequently its cost, we define a parameter f_{se} (in secs). f_{se} is the time after which a job can shrink or expand. i.e. if a job was shrunk or expanded at t secs, then it can be shrunk or expanded only after $t + f_{se}$ secs. This condition is enforced using Eq. 6.

B. Experimental Setup

1) Job Datasets: The results presented in this section are based on the job logs [41] of Intrepid [42]. Intrepid is a IBM BG/P supercomputer with a total of $40,960$ nodes and is installed at Argonne National Lab. The job trace spans over 8 months and has $68,936$ jobs. We extracted 3 subsets of 1000 successive jobs each from the trace file to conduct our experiments. These subsets will be referred to as Set1, Set2, and Set3 and the starting job ids for these subsets are 1, 10500, and 27000, respectively. To measure the performance of PARM in the wake of diverse job arrival rates, we generated several other datasets from each of these sets by multiplying the arrival times of each job by γ, where $\gamma \in [0.2 - 0.8]$. Multiplication of the arrival times with γ increases the job arrival rate without changing the distribution of job arrival times and tells us how the data center throughput would change in case the load is increased, i.e., faster job arrival rate.

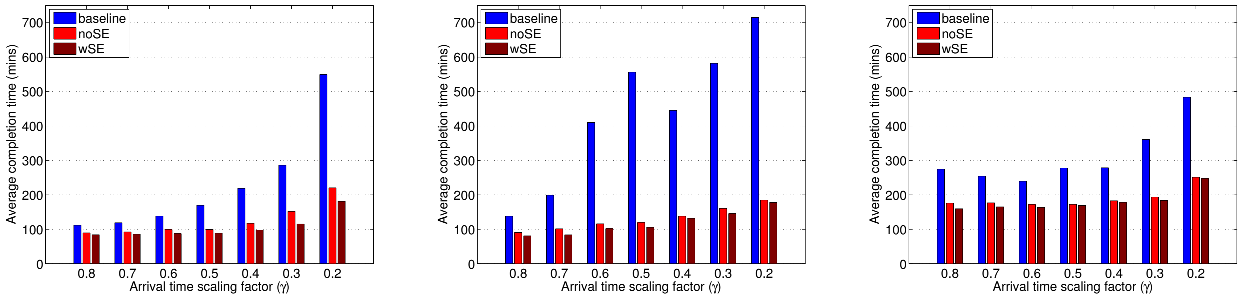
2) Application Characteristics: Since most data centers do not report power characteristics of running jobs, application characteristics, ($\sigma, T_1, A, f_l, f_h, \beta, a$ and b), of the jobs in the Intrepid logs are not known. Hence, we sampled parameter values from the range defined applications in § VI-C and randomly assign them to jobs in our datasets. Since these parameters are chosen from a diverse set of applications that range from CPU intensive to memory intensive applications, we believe them to be representative of real applications.

3) Node Range for Moldable/Malleable Jobs: Intrepid does not allow moldable/malleable jobs, and therefore logs only specify the fixed number of nodes requested by the jobs. For jobs submitted to the PARM scheduler, we consider this number as the maximum nodes that the job can be run ($max(N_j)$), and set $min(N_j) = \theta * max(N_j)$, where θ is randomly selected from the range $[0.2 - 0.6]$.

4) Power Budget and CPU Power levels: Information about power consumption of Intrepid nodes is not publicly available. Therefore, we use the power values from our testbed cluster (described in § VI-B). With 116W as the maximum power consumption per node, the maximum power consumption of $40,960$ nodes equals 4751360W. The SLURM scheduler will schedule on 40960 nodes with each node running at maximum power level. As in § VI-E, PARM uses 6 CPU power levels, $P_j = \{30, 33, 36, 44, 50, 60\}W$.

C. Performance Results

Figure 5 shows that both noSE and wSE significantly reduce average completion times compared to SLURM's baseline scheduling policy. As γ decreases from 0.8 to 0.2, the average completion time increases in all the schemes because the jobs arrive at a much faster rate and therefore have to wait in the queue for longer time before they get scheduled. However, this increase in the average completion times with both our schemes is not as significant as it is with the SLURM baseline scheme. The average completion times includes the cost of all overheads. In all the job datasets, the average overhead for shrinking and expanding the jobs was less than 1% of the time taken to execute the dataset. We controlled these costs by setting $f_{se} = 500$ secs, i.e., the scheduler waited for at least 500 secs between two successive shrink and expand operations for a job. Cost of optimizing the ILP was also very small. In the worst case, it took 15 secs to optimize an ILP,



(a) Set1

(b) Set2

(c) Set3

Fig. 5: Comparing average completion times of baseline, noSE, and wSE on several datasets.

which is negligible as compared to the frequency at which the scheduler was triggered.

Table V shows that both noSE and wSE have significantly improved average response times, with wSE outperforming noSE. Execution time in wSE includes the costs of shrinking or expanding the job (§ VII-A). Despite this overhead, wSE consistently outperforms noSE in all data sets. Better average completion times in noSE and wSE, despite having poor average execution time (Table V), as compared to SLURM is because of the fact that they can run more jobs simultaneously by intelligently selecting the number of nodes for each job. Speedups in Table V are the improvement in average completion time compared to the baseline scheme. A speedup of 5.2X is obtained in the best case. We make the following two observations in the obtained speedups:

- Higher speedups for smaller values of γ: This implies that as the job arrival rate increases relative increase in average completion time of wSE is much less than the relative increase in the average completion time of the baseline scheme.
- Smaller speedups in Set3 as compared to Set2: This is because Set3 does not have enough jobs to keep the machine fully utilized for first half of Set3.

D. Comparison with Naive Overprovisioning

To show the benefits of using a sophisticated optimization methodology for intelligent power allocation in PARM (§ IV-A), we compare PARM with a naive strategy in which all nodes in the overprovisioned system are allocated the same CPU power and the jobs are scheduled using the SLURM baseline scheduling policy. For instance, with CPU power of 30W, the naive strategy can use up to $\lfloor \frac{4751360}{30+18+38} \rfloor = 55248$ nodes. Table VI gives the speedup of wSE over the naive strategy executed with different CPU power caps. Significant speedups in Table VI clearly demonstrate that intelligent power allocation to the jobs is essential and benefits of PARM are not coming merely from overprovisioning of nodes.

E. Analyzing Tradeoff Between Fairness and Throughput

We introduced the term w_j in the objective function of the scheduler's ILP (Eq. 1) to prevent starvation of jobs with low power-aware speedup. In this section, we analyze the tradeoff between maximum completion time of any job (i.e. job fairness) and the average completion time of the jobs (i.e. data center throughput). Figure 6 shows the results from several experiments where we varied α and measured its impact on average and maximum completion times. As the value of α (Table I) increases, the maximum completion time decreases at the cost of increase in average completion time. Therefore, the parameter α can be be tuned by the data center administrator to control fairness and throughput.

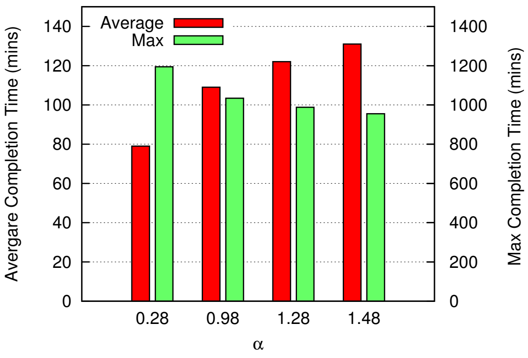


Fig. 6: Average (left axis) and maximum (right axis) completion times for Set 1 for different values of (α)

F. How Much Profile Data is Sufficient?

Number of binary variables in the PARM ILP formulation ($x_{j,n,p}$) are proportional to the number of CPU power levels of the jobs (P_j). Larger number of such power levels increase the solution space of the ILP and hence the quality of the solution. However, larger the number of variables, longer it takes to solve the ILP. As mentioned earlier in § VII-C, the cost of solving the ILP in all of our experiments was 15 secs in the worst case when $|P_j|= 6$ and the job queue $\mathcal{J}$ had 200 jobs. In this section, we show the impact of $|P_j|$ on the completion times. Figure 7 shows the average and maximum completion times of job dataset Set1($\gamma = 0.5$) as the number of CPU power levels (P_j) are increased from 2 to 8. For example, $|P_j|= 2$ means that a job can execute either at 30W or at 60W CPU power. The average and maximum completion times decreases as $|P_j|$ goes from 1 to 6 and the improvement

TABLE V: Comparing various performance metrics of baseline, wSE and noSE on various datasets

Set	Avg Resp. Time (mins)			Avg Exe. Time (mins)			Avg. Num. of Nodes			Speedup	
	baseline	wSE	noSE	baseline	wSE	noSE	baseline	wSE	noSE	wSE	noSE
1 ($\gamma = 0.5$)	90	3	6	80	84	95	453	610	601	1.91	1.70
2 ($\gamma = 0.5$)	500	34	57	57	69	89	632	714	721	5.25	4.66
3 ($\gamma = 0.5$)	217	99	88	60	73	90	520	662	665	1.65	1.61
2 ($\gamma = 0.7$)	142	12	20	57	66	83	596	656	660	2.36	1.96
3 ($\gamma = 0.7$)	194	95	86	60	73	90	488	596	599	1.54	1.43

TABLE VI: Speedup of wSE over baseline scheduler running on an overprovisioned system (i.e. the naive strategy) at different CPU power caps on Job Dataset Set2 ($\gamma = 0.5$)

CPU power cap (W)	30	40	50	60
Speedup of wSE over naive	4.32	1.86	2.33	5.25
Num. of nodes in naive strategy	55248	49493	44824	40960

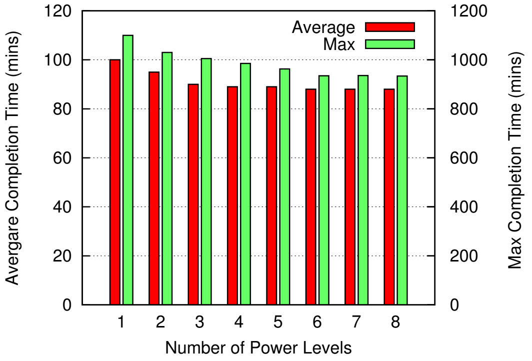


Fig. 7: Effect of increasing the number of power levels ($|P_j|$) on the average and maximum completion time of Set 1 ($\gamma = 0.5$). There is negligible improvement in performance after 6 power levels

stops as $|P_j|$ is further increased. This indicates that 6 CPU power levels were sufficient to get maximum performance from PARM for the given job datasets.

VIII. Conclusions and future work

To the best of our knowledge, PARM is the first online scheduler that uses power aware characteristics , CPU power capping and job malleability to achieve high job throughput under a strict power budget. PARM holds promise for maximizing job throughput of existing and upcoming data centers where power is a constraint. We proposed a power-aware strong scaling model that can estimates an application's power-capped performance at any scale with good accuracy. The proposed sophisticated ILP optimization methodology uses performance estimates from the model to select jobs for scheduling, and allocates CPU power caps and nodes to them. Programming models like MPI, which do not directly support job malleability can also benefit significantly from our power-aware scheduling (using noSE version of PARM). To conclude, hardware-software coordinated approaches can significantly help in driving performance-power tradeoff at exascale. Adaptive runtime systems can further increase these benefits by allowing job malleability.

Caches constitute a significant portion of the node power consumption. However, the benefits of using different levels of caches on application performance may not be proportional to their power consumption [43]. In our future work, we plan to add this additional degree of freedom to PARM, which is the ability to dynamically enable/disable caches at various levels. We also plan to provide rich support for user priorities in PARM. Thermal behavior of CPUs can significantly affect the reliability of a machine [44] as well as the cooling costs of the data center [45]. We also plan to investigate the possibility of incorporating thermal constraints along with a strict power constraint in our scheduling scheme.

References

[1] "2013 Exascale Operating and Runtime Systems," http://science.doe.gov/grants/pdf/LAB_13-02.pdf, Office of Advanced Science Computing Research (ASCR), Tech. Rep.

[2] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond DVFS: A First Look at Performance Under a Hardware-enforced Power Bound," in *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012.

[3] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13, New York, NY, USA, 2013.

[4] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. de Supinski, "Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–8.

[5] C.-H. Hsu and W.-C. Feng, "Effective Dynamic Voltage Scaling through CPU-Boundedness Detection," in *Proceedings of the 4th International Conference on Power-Aware Computer Systems*, ser. PACS'04, 2005.

[6] K. Choi, R. Soma, and M. Pedram, "Fine-grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff based on the Ratio of Off-chip Access to On-chip Computation Times," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 18–28, 2005.

[7] ——, "Dynamic Voltage and Frequency Scaling based on Workload Decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM, 2004, pp. 174–179.

[8] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: frequency-aware static timing analysis," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, Dec 2003, pp. 40–51.

[9] X. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 34–34.

[10] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 34.

[11] S. Kamil, J. Shalf, and E. Strohmaier, “Power efficiency in high performance computing,” in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–8.

[12] R. Ge, X. Feng, and K. Cameron, “Modeling and Evaluating Energy-performance Efficiency of Parallel Processing on Multicore Based Power Aware Systems,” in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–8.

[13] A. Porterfield, S. Olivier, S. Bhalachandra, and J. Prins, “Power measurement and concurrency throttling for energy reduction in openmp programs,” in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 884–891.

[14] O. Mmmel, M. Majanen, R. Basmadjian, H. Meer, A. Giesler, and W. Homberg, “Energy-aware job scheduler for high-performance computing,” *Computer Science - Research and Development*, vol. 27, no. 4, pp. 265–275, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00450-011-0189-6

[15] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, “Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[16] C.-h. Hsu and W.-c. Feng, “A power-aware run-time system for high-performance computing,” in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 1.

[17] O. Sarood, P. Miller, E. Totoni, and L. V. Kale, “‘Cool’ Load Balancing for High Performance Computing Data Centers,” in *IEEE Transactions on Computer - SI (Energy Efficient Computing)*, September 2012.

[18] B. Subramaniam and W.-c. Feng, “Towards energy-proportional computing for enterprise-class server workloads,” in *Proceedings of the ACM/SPEC international conference on International conference on performance engineering*. ACM, 2013, pp. 15–26.

[19] D. G. Feitelson and L. Rudolph, “Toward convergence in job schedulers for parallel supercomputers,” in *In Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1996, pp. 1–26.

[20] L. Kalé, “The Chare Kernel parallel programming language and system,” in *Proceedings of the International Conference on Parallel Processing*, vol. II, Aug. 1990, pp. 17–25.

[21] L. Kale, A. Arya, N. Jain, A. Langer, J. Lifflander, H. Menon, X. Ni, Y. Sun, E. Totoni, R. Venkataraman, and L. Wesolowski, “Migratable Objects + Active Messages + Adaptive Runtime = Productivity + Performance A Submission to 2012 HPC Class II Challenge,” Parallel Programming Laboratory, Tech. Rep. 12-47, November 2012.

[22] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale, “Parallel Programming with Migratable Objects: Charm++ in Practice,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’14. New York, NY, USA: ACM, 2014.

[23] L. V. Kalé, S. Kumar, and J. DeSouza, “A malleable-job system for timeshared parallel machines,” in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, May 2002.

[24] M. C. Cera, Y. Georgiou, O. Richard, N. Maillard, and P. O. A. Navaux, “Supporting malleability in parallel architectures with dynamic cpusets mapping and dynamic mpi,” in *Proceedings of the 11th international conference on Distributed computing and networking*, ser. ICDCN’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 242–257. [Online]. Available: http://dl.acm.org/citation.cfm?id=2018057.2018090

[25] K. El Maghraoui, T. Desell, B. Szymanski, and C. Varela, “Dynamic malleability in iterative mpi applications,” in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, 2007, pp. 591–598.

[26] C. Huang, O. Lawlor, and L. V. Kalé, “Adaptive MPI,” in *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, College Station, Texas, October 2003, pp. 306–322.

[27] A. B. Downey, “A Model for Speedup of Parallel Programs,” 1997.

[28] O. Sarood and L. Kale, “Efficient Cool Down of Parallel Applications,” *Workshop on Power-Aware Systems and Architectures in conjunction with International Conference on Parallel Processing*, 2012.

[29] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, “Fast: Frequency-aware Static Timing Analysis,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 5, no. 1, pp. 200–224, 2006.

[30] J. M. Cebrian, J. L. Aragón, J. M. García, P. Petoumenos, and S. Kaxiras, “Efficient Microarchitecture Policies for Accurately Adapting to Power Constraints,” in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–12.

[31] R. Graybill and R. Melhem, *Power aware computing*. Kluwer Academic Publishers, 2002.

[32] X. Chen, C. Xu, and R. Dick, “Memory Access Aware on-line Voltage Control for Performance and Energy Optimization,” in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, 2010, pp. 365–372.

[33] I. Karlin, A. Bhatele, J. Keasler, B. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang *et al.*, “Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application,” in *International Parallel and Distributed Processing Symposium*, 2013.

[34] A. Langer, J. Lifflander, P. Miller, K.-C. Pan, L. V. Kale, and P. Ricker, “Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement,” in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*. IEEE, 2012, pp. 100–107.

[35] A. Langer, “An Optimal Distributed Load Balancing Algorithm for Homogeneous Work Units,” in *Proceedings of the 28th ACM International Conference on Supercomputing*, ser. ICS ’14. New York, NY, USA: ACM, 2014, pp. 165–165. [Online]. Available: http://doi.acm.org/10.1145/2597652.2600108

[36] Intel, “Intel-64 and IA-32 Architectures Software Developer’s Manual , Volume 3A and 3B: System Programming Guide, 2011.”

[37] M. A. Jette, A. B. Yoo, and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[38] A. Lucero, “Slurm Simulator,” http://www.bsc.es/marenostrum-support-services/services/slurm-simulator, Tech. Rep.

[39] “Top500 supercomputing sites,” http://top500.org, 2013.

[40] ANL, “Running Jobs on BG/P systems,” https://www.alcf.anl.gov/user-guides/bgp-running-jobs#boot-time.

[41] “Parallel Workloads Archive,” http://www.cs.huji.ac.il/labs/parallel/workload/, Tech. Rep.

[42] Intrepid. [Online]. Available: http://www.top500.org/system/176322

[43] E. Totoni, J. Torrellas, and L. V. Kale, “Using an adaptive hpc runtime system to reconfigure the cache hierarchy,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’14. New York, NY, USA: ACM, 2014.

[44] O. Sarood, E. Meneses, and L. V. Kale, “A ’cool’ way of improving the reliability of hpc machines,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’13. New York, NY, USA: ACM, 2013, pp. 58:1–58:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503228

[45] O. Sarood and L. V. Kalé, “A ‘cool’ load balancer for parallel applications,” in *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA, November 2011.