PoDD: Power-capping Dependent Distributed Applications

Huazhe Zhang
huazhe@cs.uchicago.edu
University of Chicago

Henry Hoffmann
hankhoffmann@cs.uchicago.edu
University of Chicago

ABSTRACT

Power budgeting (or capping) has become essential for large-scale computing installations. Meanwhile, as these systems scale out, they can concurrently execute dependent applications that were previously processed serially. Such application *coupling* reduces IO traffic and overall time to completion as the applications now communicate at runtime instead of through disk. Coupled applications are predicted to be a major workload for future *exascale* supercomputers; *e.g.,* scientific simulations will execute concurrently with *in situ* analysis. One critical challenge for power budgeting systems is implementing power capping for coupled applications while still achieving high performance. Existing approaches on power capping coupled workloads, however, have major limitations including: (1) *poor practicality*, due to dependence on offline application profiling; and (2) *limited optimization opportunity*, as they consider power reallocation on a strictly global level (from node-to-node), without considering node-level optimization opportunities.

To overcome these limitations, we propose *PoDD*, a hierarchical, distributed power management system for coupled applications. *PoDD* uses classifiers and online model building to determine optimal power and performance tradeoffs without offline profiling or application instrumentation. We implement it on a 49-node cluster and compare it to SLURM, a state-of-the-art job scheduler that considers power, but not coupling, and *PowerShift* , a power capping system for coupled applications without node-level optimization. *PoDD* improves mean performance over SLURM by 14–22% and over *PowerShift* by 11–13%. Finally, *PoDD* is resilient to tail behavior and system noise, improving performance in noisy environments by 44% on average compared to even power distribution.

CCS CONCEPTS

• **Hardware → Enterprise level and data centers power issues**.

KEYWORDS

Machine Learning, Power Management, Adaptive Systems

ACM Reference Format:
Huazhe Zhang and Henry Hoffmann. 2019. *PoDD*: Power-capping Dependent Distributed Applications. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3295500.3356174

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356174

1 INTRODUCTION

One of the most exciting opportunities of emerging exascale systems is that abundant computing resources will allow a promising new execution model for previously sequential dependent jobs. Specifically, applications can be *coupled* so that they run concurrently and communicate through the network rather than running them sequentially and communicating through the file system [2, 5, 7, 31, 57]. For example, scientific simulations now could be coupled with *in situ* data analysis or visualization [2, 7]. The US Department of Energy (DoE) has declared resource management for coupled applications a key challenge for exascale computing [5, 57]. The importance of coupled applications is further demonstrated by recent work on pairing the widely used LAMMPS simulation code with *in situ* analysis [37, 38].

Realizing these opportunities requires addressing challenges raised by exascale, including the related concerns of power constraints and node-level complexity [57]. At the processor level, computing density is out-pacing cooling capacity: if all transistors on a chip were used simultaneously, the processor would generate more heat than can be dissipated and damage itself [19, 58]. At the same time, experts predict that exascale supercomputers will need to operate in a 20–80 MW power budget [5].

To address these power concerns, hardware manufacturers have made individual nodes increasingly *configurable*. Almost all processors now support power management either through exposing voltage and frequency settings to software or allowing software to explicitly set power limits [11]. Additionally, aggressive power gating means that performance and power tradeoffs can be further tuned by idling (or choosing not to use) node-level resources [62]. For exascale systems, the challenge is determining the node-level configurations that respect the system-wide power budgets while delivering the maximum possible performance.

While a great deal of research explores power control for large-scale systems, most does not account for dependent or coupled workloads. For instance, some research increases overall system throughput by better utilizing hardware resources [29], reallocating unused power [54], balancing computing and IO power [53], or mitigating the impact of manufacturing variability [27]. These studies resolve important issues (unbalanced allocation and phase-adaptation) for power capping distributed systems, but all assume independent workloads. Dependent/coupled workloads, however, follow a fundamentally different performance principal. Specifically, for coupled applications the overall speed is determined by the slowest application. Therefore, under a global resource constraint—like a system-level power budget—ideal coupling-aware power management speeds up the slowest application until each is running at the

same speed [14]. None of the works mentioned above is aware of the relative speed between coupled applications nor able to shift power from the faster application to the slower one.

The only work we know of addressing power management for coupled applications is *PowerShift* [63]. While *PowerShift* represents a family of related algorithms for shifting power between coupled applications, its distributed, dynamic approach delivers higher performance under power caps than approaches that are not coupling-aware. Unfortunately, PowerShift has two major drawbacks limiting both usability and achievable performance. First, it requires users to provide offline profiles of the power and performance tradeoffs for the coupled applications. Thus, both applications will have to be run many times before being coupled together under the power cap. Second, *PowerShift* only manages a single node-level resource: processor voltage and frequency. However, several prior works show that coordinating voltage and frequency with other node-level resources (like socket, core, and memory usage) leads to much higher performance for the same power budget [8, 10, 15, 16, 20, 35, 48, 49, 52, 59, 62].

To overcome the limitations of prior approaches, we propose *PoDD*, a dynamic, hierarchical power management system. *PoDD* requires no offline profiling data or code instrumentation. *PoDD* operates in three phases: (1) classifying applications based on their optimal resource usage, (2) determining optimal power/performance tradeoffs for the coupled applications, and (3) dynamically adjusting power usage in an efficient, distributed manner to account for application phases, system noise, and tail behavior. *PoDD* amortizes overhead by dedicating one node to learning; all application nodes send performance counter information to the dedicated node, which coalesces the data and returns classification results for the coupled applications in the couple.

We implement *PoDD* on a 49 node system and evaluate it against 4 widely-used/state-of-the-art power control systems: *Fair*, *SLURM* [54], *Optimal Static Power Allocation* [63] and *PowerShift* [63]. We normalize all results to *Fair* power allocation, which simply divides the power budget equally among all nodes. We find that:

- *SLURM* improves mean performance by 6%.
- *Optimal Static Allocation* improves mean performance by 8%.
- *PowerShift* improves mean performance by 12%.
- *PoDD* improves mean performance by 28%.
- *PoDD* mitigates tail effects and system noise, improving mean performance by 31% and 44%, reducing standard deviation of the runtime of each node by 40.4% and 41.3% under these conditions.
- *PoDD* is topology-oblivious—it works well whether coupled applications are physically separate or co-located.

Furthermore, our scalability emulation and analysis estimates that *PoDD* can scale up to thousands, or even tens of thousands of nodes.

In summary, many prior works shift power among nodes in a distributed environment [27, 29, 54, 63], but these approaches all assume simple models of node-level power management—typically that node-level power is adjusted only through dynamic voltage and frequency scaling (DVFS). Concurrent research on node-level power management shows significant performance gains over DVFS by coordinating more node-level resources (e.g., DVFS plus socket, core, hyperthreads, memory usage) [62]. This paper is the first work to propose a framework for combining distributed power shifting with node-level power optimization. Furthermore, this paper demonstrates the benefits of the proposed approach for the important class of coupled applications. More specifically, *PoDD* addresses the three intertwined challenges of (1) optimizing node-level performance for a power budget, (2) distributing a system-level power budget, and (3) dynamically adjusting power to achieve high performance for coupled applications. Significantly for adoption by scientists, *PoDD*'s solution requires minimal input from users.

2 BACKGROUND AND MOTIVATION

Many distributed power capping approaches have been proposed. In this section, we briefly introduce several such prior systems, which we will use to evaluate *PoDD*. Then, we point out two major limitations of prior work and describe how *PoDD* overcomes them. Section 5 contains a full treatment of related work.

2.1 Prior Power Capping Approaches

We survey four approaches from the literature. The first two are widely-used, real-world power control systems: *Fair* and *SLURM* [54]. Next, we review two system designs within the *PowerShift* family, the first power capping system designed for coupled applications: *PowerShift-S* is a static approach, *PowerShift-D* is a dynamic approach [63].

Fair power allocation refers to evenly dividing the whole system power cap among each node without knowledge of what applications are running. Moreover, *Fair* allocation has no mechanism to make any runtime power adjustments. *Fair* is widely used as the default power capping model in data centers, supercomputers, or other large-scale systems. This heuristic is simple to implement and works as a one-size-fits-all approach that is workload independent. *Fair* allocation, however, cannot tune power allocation for different workloads. We use *Fair* as our experimental baseline.

SLURM is a state-of-art job scheduler for large-scale distributed systems with an adaptive power capping mechanism. *SLURM* starts like *Fair*—with power divided evenly across all nodes—and monitors runtime power consumption to redistribute power using a simple heuristic. Specifically, *SLURM* divides nodes into two groups: those using less power than their assigned cap and those operating near their cap. *SLURM* dynamically reduces the power budget of the nodes in the first group and splits the excess power among nodes in the second group. This heuristic takes advantage of unused power to increase overall throughput of independent applications, but is often sub-optimal with coupled applications where overall performance is determined by the slowest application.

PowerShift is a family of power management solutions designed for coupled application workloads [63]. All *PowerShift* approaches require offline profiles of each application's performance and power tradeoffs. Furthermore, *PowerShift* only manages node-level DVFS and does not consider any other node-level resources.

We are concerned with two variations of *PowerShift*: a static (*PowerShift-S*) and a dynamic (*PowerShift-D*) approach. *PowerShift-S* takes advantage of offline power and performance profiles to determine the optimal static power allocation between coupled applications. This approach, however, does not make any dynamic power adjustment, so it cannot adapt to runtime changes. *PowerShift-D* is a dynamic, decentralized power manager with two key features: a power pool, and power priority groups. Unlike *SLURM*—which

only shifts power from nodes that are not using their full budget—*PowerShift-D* has an additional priority group allowing it to shift power from a fast application operating at its budget to a slow application also at its budget.

2.2 Major Limitations and Solutions

PowerShift, to our knowledge, is the first work to address the challenge of dependent distributed applications; however, it has two major limitations:

- Its dependence on offline profiles limits practicality.
- Its hardware-only (DVFS-only) power capping at the node-level limits opportunities for additional performance.

We note that this last limitation appears to be common among many distributed power capping approaches. In the following sections, we will demonstrate each of the challenges and motivate its solution.

2.2.1 Online Power Performance Profiling. Relying on offline profiles has several drawbacks: (1) they may simply not be available, (2) collecting them can be very costly, especially in large scale systems and (3) for applications that dramatically change behavior when given different input and/or arguments, offline profiles are misleading. Thus, there is a need for a practical approach to automatically constructing these profiles online.

Since power capping systems operate in a feedback control loop—constantly monitoring performance and adjusting resources or processor frequency to maintain the cap—it is natural to collect power and performance data online and build a predictive model. The optimization tradeoff is how many data points to collect—fewer data points reduces overhead, but possibly at a cost of higher error in the predictions. Researchers have proposed several techniques for building predictive models of system power and performance. Some collect hardware performance counters and make predictions based on an empirical model [56]. Several approaches apply advanced learning techniques including multi-linear regression [46], probabilistic graphical models [42, 44], matrix completion [12, 13, 17], and even deep learning [39]. In our case, the nodes running the same application are power capped evenly, so our problem's search space is very small. For example, in a cluster of nodes with 120 W TDP and 50 W minimum power limit, the most power space we need the model to cover is 70 W. In practice the space is even smaller as one application will be faster and cover the lower end of the range, while the other application will be slower and want more power. Given these observations, we find a binary search model works well as it needs only a few observation data points and provides a maximum power error of 2 W (the largest difference between desired and measured power) and has minimal overhead because of the simplicity. We will demonstrate and evaluate this binary search model in detail in later sections.

2.2.2 Optimizing Node-level Power Capping. We now highlight the benefits of incorporating more complex node-level power management over simply deploying hardware power capping. Because different parallel applications may favor different system configurations each application has its unique requirement for resources, including computing, memory, and IO. Fig. 1 shows the performance and power tradeoff spaces of the *galaxy* scientific simulation [55], with each marker representing one run of *galaxy* with a different node-level resource configuration. The configuration space consists of (1) whether or not the node uses dual sockets, hyperthreads, dual memory controllers, and (2) per-node power caps from 100 W to 240 W in 20W increments. Thus, the total number of configurations in this example is 64 (2*2*2*8). The system's default configuration is to use all available resources—dual sockets, hyperthreading, and dual memory controllers. As shown in the figure, *galaxy* clearly favors a non-default resource allocation (single socket, no hyperthreading, with dual memory controllers). The optimal configuration achieves a 58% performance improvement under the same power cap. In such cases, only setting the hardware power cap would result in sub-optimal configuration and great performance loss.

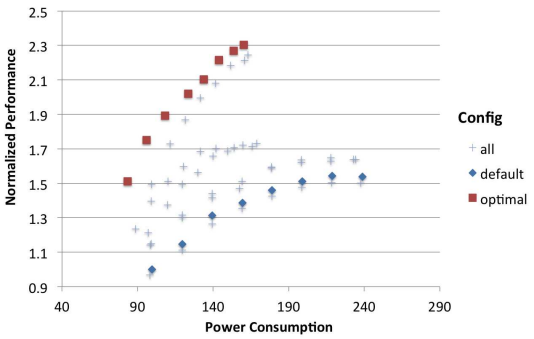

Figure 1: Performance of different node configurations.

Achieving better performance calls for more complex node-level power capping to coordinate multiple resources. The *PUPiL* power management system demonstrates the benefit of this coordinated approach over common approaches that only consider processor voltage and frequency [62]. One key constraint for PUPiL, however, is that it requires users to add instrumentation to the application code so that it can measure performance. Inspired by the performance potential demonstrated both in PUPiL and in Fig. 1, we develop a system that coordinates multiple node-level resources without code instrumentation. Specifically, we combine a machine learning classifier with hardware capping to enforce the power cap. The classifier makes decisions about which configuration is more suitable for the current application by monitoring hardware performance counters online. Polling the performance counters is lightweight and we design a classifier model to use a very small number of features to reduce computational complexity. In this way, the proposed design not only eliminates code instrumentation, it also keeps the classifier's overhead low and suitable for online decision-making. More detailed description and evaluation of the machine learning classifier follows in the next sections.

3 *PODD* DESIGN

PoDD is a hierarchical, distributed, dynamic power management system that handles the unique challenges of maximizing performance for dependent application workloads. For naming simplicity we refer to the two parts of an application couple as the front- and back-ends. The front-end produces data that is consumed by the back-end, so the couple represents a short pipeline. The couple's overall performance will thus be determined by the slowest end, and an ideal resource management system would apportion resources

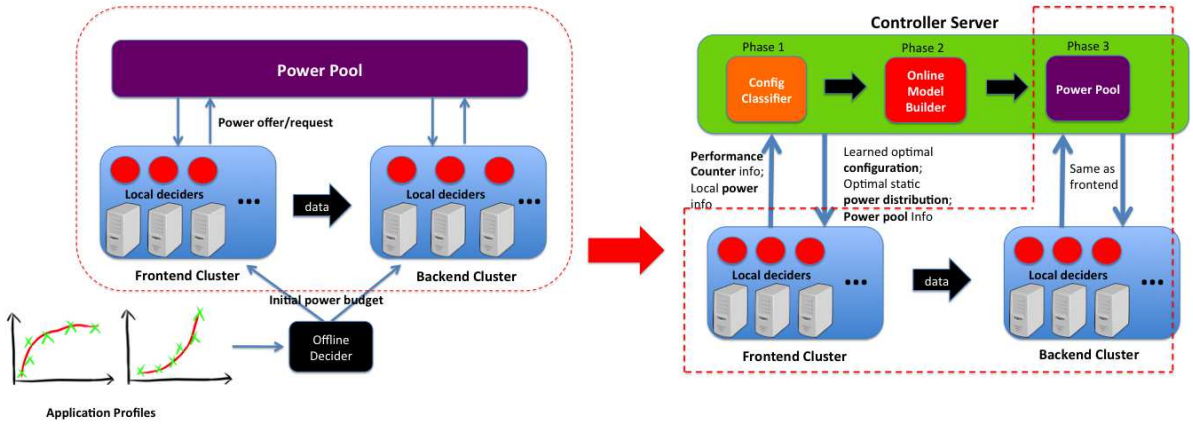

Figure 2: Overview and comparison of *PowerShift-D* (left) and *PoDD* (right).

such that each end takes the same time. Under a power cap, then, the power management system should move power from the faster application to the slower until both take the same time. Prior work demonstrates that such a power distribution is optimal [14, 63].

PoDD uses three phases to achieve a near-optimal power allocation. In the first phase, *PoDD* collects hardware performance counter data from each node and passes that data to a *configuration classifier* to predict the optimal node-level *configuration* for the front- and back-ends, respectively. Here a configuration represents a usage of node-level resources (other than power, which will be managed separately); *e.g.,* a configuration might be the use of two memory controllers, 1 socket and hyperthreads. In the second phase, *PoDD* builds models of the power and performance tradeoffs for the front- and back-ends. This second phase determines an optimal static assignment of power to the front- and back-ends, and thus augments the optimal configurations from the first phase with power information. At the end of this phase, the front- and back-end nodes may have different power allocations and different node-level resources in use. The third and final phase performs dynamic power shifting by coordinating node- with system-level power management. In this final phase, power will dynamically shift from fast to slow nodes with the goal of having the front- and back-end complete at the same time. We note that *PoDD*'s first and third phases (classifying node-level needs and coordinating between node and system level) function outside the domain of coupled applications and are likely more broadly useful.

Fig. 2 shows the *PoDD*'s overall design and compares it to *PowerShift*. As shown in the figure, the key difference is that *PoDD* automatically determines optimal node-level configurations and static power-performance tradeoffs so that users do not need to perform any profiling or code instrumentation. In other words, *PoDD* automatically collects the data and constructs the models required to use *PowerShift's* dynamic power management scheme. We now discuss each phase and its key component in detail.

3.1 Phase 1: Configuration Classification

To achieve better overall performance, *PoDD* incorporates more complex node-level power capping than prior approaches, which simply rely on hardware to manage processor voltage and frequency. Our two goals for this phase are: (1) classifying application performance without code instrumentation and (2) minimizing overhead.

Table 1: Overview of performance counters monitored.

Performance Counter	Description
EXEC	Instructions per nominal CPU cycle
IPC	Instructions per cycle
FREQ	Frequency relative to nominal CPU frequency
AFREQ	FREQ excluding when CPU is sleeping
L3MISS	L3 cache line miss
L2MISS	L2 cache line miss
L3MPI	L3 cache line miss per instruction
L2MPI	L2 cache line miss per instruction
READ	Memory read traffic
WRITE	Memory write traffic
INST	Instruction retired
PhysIPC%	IPC relative to maximum IPC
INSTnom%	Relative instructions per nominal cycle
TotalQPIout	QPI data traffic estimation
QPItoMC	Ratio of QPI traffic to memory traffic

To achieve these goals, *PoDD* monitors hardware performance counters at runtime and uses a machine learning classifier to predict the most suitable configuration for the current workload. This approach is advantageous because (1) the performance counters can be easily read without code instrumentation and (2) the application can be classified after a brief period, reducing overhead. We use Intel's Performance Counter Monitor (PCM) to collect performance counter data at runtime. Table 1 shows the overview of system-level performance counters that we monitor. The counter information is normalized to produce per instruction rates (if needed). For example, the READ and WRITE counts are translated into memory traffic per instruction by dividing the measured data by the total instructions retired during the monitor phase.

Each counter serves as one feature for the classifier. They further go through a series of pre-learning procedures including: noise filtering, data standardization, PCA (principal component analysis), and feature selection. At that point they are fed into the classifier to predict the best configuration for the current application.

Two-level classifier: A key design choice is to have a two-level classifier that predicts computing and memory resources separately

Table 2: Classifier models explored.

Classifier model	Description
SVM-linear	Support vector machine with linear kernel
SVM-poly	Support vector machine with polynomial kernel
SVM-rbf	Support vector machine with rbf kernel
KNN	K-nearest neighbor classifier
RF	Random forest, using a bunch of decision trees for classification
ET	Extra-tree classifier, a variation of RF, using extreme randomized decision trees
AB	AdaBoost, boosted with decision trees
LR	Logistic regression classifier

instead of treating the problem as a multi-class classification. This choice means that, instead of predicting all configuration preferences (socket allocation, use of hyperthreads, memory controller allocation) at once, we break predication into two parts: (1) predicting computing resource preferences (socket allocation and hyperthread usage) and (2) predicting memory resource preferences.

We choose this two-level scheme because classifying compute and memory needs simultaneously makes the problem harder. The difficulty arises in part from the fact that memory and compute resources are not completely independent. For example, low IPC may result from either poor compute resource allocation or from an application that constantly misses in the cache. In the first case, IPC will improve by allocating more compute resources. In the second case, more computing resources will consume more power, but without increasing performance.

For completeness, we test a multi-class classifier that predicts both memory and compute needs. That approach, however, has much worse accuracy. The multi-class classifier achieves 80% recall with less than 70% precision. In comparison, *PoDD*'s multi-level classifier achieves 95% recall and 74% precision.

Model selection: We compare eight popular classifier models, shown in Table 2. In all cases, we perform classification after preprocessing the hardware PCM data by performing noise filtering, data standardization, PCA (principal component analysis), and feature selection. All models are tuned by hyper-parameter search. The key evaluation criteria here is that *high recall rate is a must, and the higher precision the better.* In other words, our system can tolerate false positives but is more sensitive to false negatives. The intuition is that the system will use dynamic feedback, so if it selects a sub-optimal configuration (a false positive) the low performance will be detected and corrected (at the penalty of briefly running slower than necessary). However, if we have false negatives, there will be an optimal configuration which the system will never use, limiting the potential performance gains.

Thus, we tune all models with the goal of 95% recall rate if possible, and then we pick the model with the highest precision. Fig. 3 shows the recall and precision for both predicting computing and memory resources using the different learning models from Table 2. "C" stands for computing resources and "M" stands for memory resources. As the chart shows, svm-rbf delivers above 95% recall with the highest precision for both resource classes.

Feature transformation and selection: Fig. 4 shows the result of the Principal Component Analysis (PCA) procedure. PCA is a common orthogonal transformation approach to eliminate correlation among raw feature data. This process is important when we use hardware performance counters as input, because there are quite a few correlated counters; *e.g.,* EXEC, FREQ, and AFREQ; L3Miss and READ/WRITE. Another benefit of PCA is that—combined with

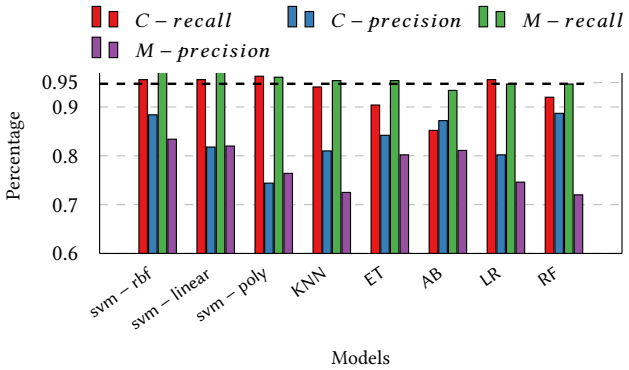

Figure 3: Recall and precision of 8 learners for classifying computing and memory resources.

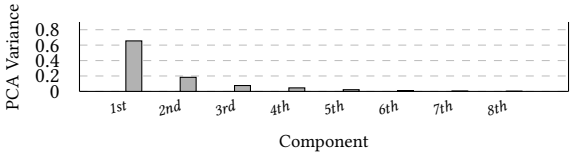

Figure 4: Eight features explain more than 99.8% of variance.

Table 3: Accuracy comparison of using top X components.

Top X components	Recall	Precision
1	0.912	0.861
2	0.941	0.727
3	0.963	0.824
4	0.963	0.852
5	0.963	0.885
6	0.948	0.860
7	0.941	0.848
8	0.941	0.848

feature selection—it serves to reduce the number of noisy features fed to the classifier and thus both improves predictions and significantly reduces computation overhead compared to simply using all features. In general, the higher variance carried by the top few principal components, the better. In this case, the top 8 principal components account for more than 99.8% variance, and the top 5 account for more than 98.1%. Table 3 shows the cross-validated average recall and precision using a different number of top X components as features. The performance increases until the component number reaches 5. This data implies that components beyond the fifth are very likely to add noisy or redundant features to the model. Therefore, in *PoDD*, we use the top 5 components as input features to minimize overhead and increase accuracy.

In summary, during the classification process, hardware performance counter data is collected and normalized. This data is sent to the learning node, which collects all node-level data for both front and back end applications. The data is coalesced and used to classify each application in terms of the optimal configuration of compute and memory resources. These predicted optimal configurations are then sent to all individual nodes. After this phase, each node should have an optimal resource configuration, and *PoDD* transitions to the next phase: building power/performance tradeoff models.

3.2 Phase 2: Online Model Building

One of *PoDD*'s key design features is finding the optimal power distribution between front- and back-ends online. This process

requires modeling how both the front- and back-end performance changes with changing power allocations. These models determine a power distribution between front- and back-ends such that each completes its work at close to the same time.

Two key factors make online model building natural with *PoDD*:

- It is easy to collect the necessary data points.
- In coupled applications it is easy to detect the synchronization points between front- and back-end applications and the time taken to arrive at those points is a good performance indicator.

Similar to the classifier, we want the power/performance model builder to have high accuracy and low overhead. For this component, however, accuracy means the predicted performance under some power cap should be fairly close to the true value. Low overhead means both fast convergence and low computational overhead to construct the model. We compare three different methods of online model building and test their accuracy and overhead (the details are in Section 4.6). Of these three, the binary search approach has the highest accuracy and close to the lowest overhead because the range of power distributions *PoDD* has to explore online is quite small. There is an underlying assumption here, however, which is that after classification determines optimal node-level resource usage, the power/performance tradeoff space is convex. In practice, we find that classification's high recall and dynamic feedback eliminate local optima.

Algorithm 1 details the binary search algorithm as implemented for this specific problem. All the power caps mentioned are per-node power caps, and the two parameters to be set by users are the power resolution and performance resolution. Power resolution is a threshold for how close the system should get to the desired power budget. Performance resolution is a threshold for determining when the front- and back-end applications are considered to be running at the "same" speed. Due to system noise and variance we never expect the front- and back-ends to actually complete at exactly the same time. In our case, we use 2 W as the power resolution, since lower numbers do not impact performance; for performance resolution, we use 2%, meaning that if the coupled applications reach a synchronization point within 2% of each other, we consider them to be running at same speed.

Extension beyond two coupled applications: *PoDD* can be easily extended beyond just two dependent applications. Both phase 1, configuration classification, and phase 3, dynamic power shifting, work independently of the number of applications. We, thus, briefly describe the high-level idea of extending phase 2, the online model builder, to handle N ($N > 2$) applications. Each application starts with the same (even) power cap and with same range of possible power caps (denoted by P). The optimal overall performance is always between the current highest and lowest performance. Every iteration, we find the applications with highest and lowest performance. For the highest performance one, we eliminate any power above its current power cap from its possible power cap range. For the lowest performance one, we eliminate any power below its current power cap from its possible power cap range. We then reset these two extreme applications to power caps in the middle of their possible power range. This procedure is repeated until the system is within the power resolution. So, each iteration throws away at least half the possible power range from at least one application. For each application, this process lasts at most $\log(P)$ steps, so the overall complexity is $N\log(P)$. As the number of applications, N, gets larger, the number of iterations needed grows linearly. Nevertheless, we expect N to be quite small for now and the near future.

Algorithm 1 Binary Search Model Algorithm

Require: Power cap P, maximum power limit P_{max}, minimum power limit P_{min}
Require: Power resolution R_{pwr}, performance resolution R_{perf}
Require: Power cap for frontend and backend P_{front}, P_{back}
Require: Current performance and power for frontend and backend applications: $(power_f, perf_f)$,$(power_b, perf_b)$
Feedback $(power_f, perf_f)$,$(power_b, perf_b)$, when $P_{front} = P_{end} = P$
$\delta_{power} = MIN(P_{max} - P, P - P_{min})/2$ ▷ max shiftable power
$r = perf_f > perf_b ? 1 : -1$ ▷ r reverses power shifting
while $\delta_{power} > R_{pwr}$ and $\delta_{perf} > R_{perf}$ **do**
  $P_{front} = P_{front} - r * delta_{power}$
  $P_{back} = P_{back} + r * delta_{power}$
  Apply power caps P_{front} and P_{back}.
  Get feedback: $(power_f, perf_f)$, $(power_b, perf_b)$.
  $\delta_{power} = \delta_{power}/2$ ▷ decrease search range by 2
  $\delta_{perf} = |perf_f - perf_b|/MAX(perf_f, perf_b)$
  $r = perf_f > perf_b ? 1 : -1$
Return optimal static power distribution (P_{front}, P_{back})

3.3 Phase 3: Dynamic Power Shifting

In this phase, *PoDD* dynamically tunes the power allocation. At this point, power can be shifted between front- and back-end nodes, or between nodes running the same application to account for tail-latency or system noise. While *PoDD*'s first two phases represent original work, this phase builds off of *PowerShift's* distributed, dynamic power management infrastructure. Specifically, *PoDD* integrates *PowerShift's*: *power priority grouping* and *power pool*, but gets optimal power distribution from *the online model* of phase 2, instead of *offline* profiles. Furthermore, each local node now executes with a resource configuration learned by Phase 1's classifier instead of the node's default configuration.

Power priorities: There are 3 priority groups:

1 Nodes operating below their assigned power cap.
2 Nodes operating near the power cap for which the online profile predicts additional power will *not* improve couple performance.
3 Nodes operating near the power cap for which the online profile predicts that additional power *will* improve couple performance.

Power is always shifted from lower priority groups to higher ones—Group 3 has the highest power priority, then group 2, lastly group 1. In other words, nodes operating near their power cap and running the slower application have priority for power. This grouping mechanism allows *PoDD* to reallocate power even when both front- and back-end nodes are operating near their power limit, which is the key mechanism for optimizing dependent applications.

Power pool: This shared data structure coordinates power shifting between local nodes. It keeps track of the minimum information needed: how much power is in the pool (unused power) and how many nodes are in *Group 3*. This structure is the key to enforce a strict, system-wide power limit. It keeps the invariant that any node giving up power to the pool must first lower its local power cap and any node taking power from the pool can only increase its local power cap after decreasing the power from the shared pool.

Each local node operates in a classic control loop: observing its environment, deciding on a response, and acting to implement

its decision [23]. In the observation phase, it collects local power consumption. In the decision phase, it first places itself into one of the three *power priority groups*. It then sends the power and priority group information to the shared power pool. Upon receiving the response from the power pool, it acts to change its local power allocation based on the model produced by the *configuration classifier*. This control loop happens every 2 seconds for the nodes in group 1 and 2, and 1 second for the nodes in group 3, to quickly shift power back to group 3. This process of observing power, interacting with the pool and adjusting the local power cap is repeated continually as the coupled application executes. In this manner *PoDD* constantly fine tunes its power allocation and can adapt to phases within an application, application tail latency, and system noise.

3.4 Advantages and Limitations

PoDD adds significant adaptive capability—and therefore flexibility—to power management. We briefly highlight the types of behavior which *PoDD* can and cannot handle. First, *PoDD* can adapt to behavior that is:

- *Input dependent:* Because classification and model building happen online, *PoDD* is robust in the face of applications whose behavior varies per input.
- *Couple dependent:* Because profiling is online, *PoDD* works even if applications change behavior based on what application they are coupled with. For example, it might be the case that one back-end is memory bound when coupled with one front-end, but computing bound with another.
- *Stochastic:* Some applications will have significant tail behavior. *PoDD*'s ability to dynamically shift power allows it to respond to applications that have poor load-balancing, are affected by system noise, or have any other issue that results in significant tail performance.
- *Dynamically varying:* Phase 3 performs dynamic power shifting and this process allows *PoDD* to adjust to applications that go through phases, especially phases with higher and lower power requirements.

Of course, *PoDD* is not a panacea and there are some situations to which it will not be able to adapt, including behavior that:

- *Varies faster than the classifier can act.* This is a common pitfall of all adaptation mechanisms: if the system being managed changes behavior faster than the adaptation mechanism can react, then the adaptations will never catch up.
- *Changes significantly after the initial classification phase.* In the current *PoDD* design, it never revisits classification after phase 1, which may cause sub-optimal behavior if an application spends several iterations as memory bound and then transitions to compute bound behavior later. We have not observed this behavior in any of our test applications. If it later becomes prominent, *PoDD* could account for it by dynamically monitoring whether the measured performance is close to the predicted performance and then reverting back to phase 1 if they ever diverge significantly.

4 EXPERIMENTAL EVALUATION

This section introduces our experimental setup and compares the performance efficiency of *PoDD* to 4 prior works: *Fair*, *SLURM*, *PowerShift-S* (which statically allocates power based on offline profiles), and *PowerShift-D* (which dynamically allocates power, but still requires offline profiling).

Table 4: Comparison of performance under power caps.

Power Cap	SLURM	PowerShift-S	PowerShift-D	*PoDD*
5760W	1.08	1.11	1.16	1.33
6720W	1.07	1.07	1.13	1.29
7680W	1.06	1.08	1.12	1.31
8640W	1.05	1.09	1.12	1.27
9600W	1.04	1.03	1.07	1.21
Har. Mean	**1.06**	**1.08**	**1.12**	**1.28**

4.1 Experimental Setup

Benchmarks There are 8 individual applications we use for evaluation, including 3 scientific simulations: `cluster` and `galaxy` from the cosmological simulation suite Gadget 2.0 [55]and the hydrodynamic simulation benchmark `lulesh` [30]. We use 3 different applications as proxies for in situ analysis and visualization: a scalable visualization application `VisIt` [9], a data compression benchmark `pigz` [1], and an unsupervised learning algorithm `kmeans` [6]. We also explore 2 Spark [60] applications from SparkBench [34]: `SparkKmeans` and `SparkSQL`. These benchmarks feature important workloads in distributed systems, with different resource needs, including compute-intensive, memory-intensive, and IO-intensive applications. We then create pairs of these 8 application to represent the emerging workloads of coupled applications. All coupled pairs are in the form of (front-back), where the back-end application takes the output of front-end application as input. Overall, we have 15 pairs: `cluster-VisIt, cluster-kmeans, cluster-pigz, cluster-SparkKmeans, cluster-SparkSQL, galaxy-VisIt, galaxy-kmeans, galaxy-pigz, galaxy-SparkKmeans, galaxy-SparkSQL, lulesh-VisIt, lulesh- kmeans, lulesh-pigz, luelsh-Sparkkmeans, SparkSQL-SparkKmeans`. All applications are launched with 48 threads per node, which is the maximum number of virtual cores in our test servers. Additionally, all applications are relatively long-running, taking at least a few minutes up to several hours.

Platform Our test system is a 49-node cluster. Each node is a dual-socket server, with 2 Intel Skylake Xeon Gold 6126 CPUs. The nominal clockspeed is 2.60 GHz. Each node has 256 GB of RAM divided between dual memory controllers. All nodes support Intel RAPL technology for enforcing power caps in hardware. Each processor has 12 physical cores, with hyperthreading, giving a total of 48 virtual cores across both sockets. These nodes are connected with 32-port software-defined 10 GigE switches.

Metrics We use $1/runtime$ as our performance metric. All performance numbers are normalized to *Fair*.

4.2 Performance

PoDD and 4 other power management systems are evaluated with our couples across 5 different system power budgets from 5760W to 9600W. While the system-level power budget is enforced by all 5 approaches—*i.e.,* the sum of 48 node-level power is always less than or equal to the system-level budget—the performance varies.

Table 4 summarizes the overall harmonic mean performance normalized to *Fair* for each of the evaluated power managers under different power caps. All approaches outperform *Fair*. *SLURM*

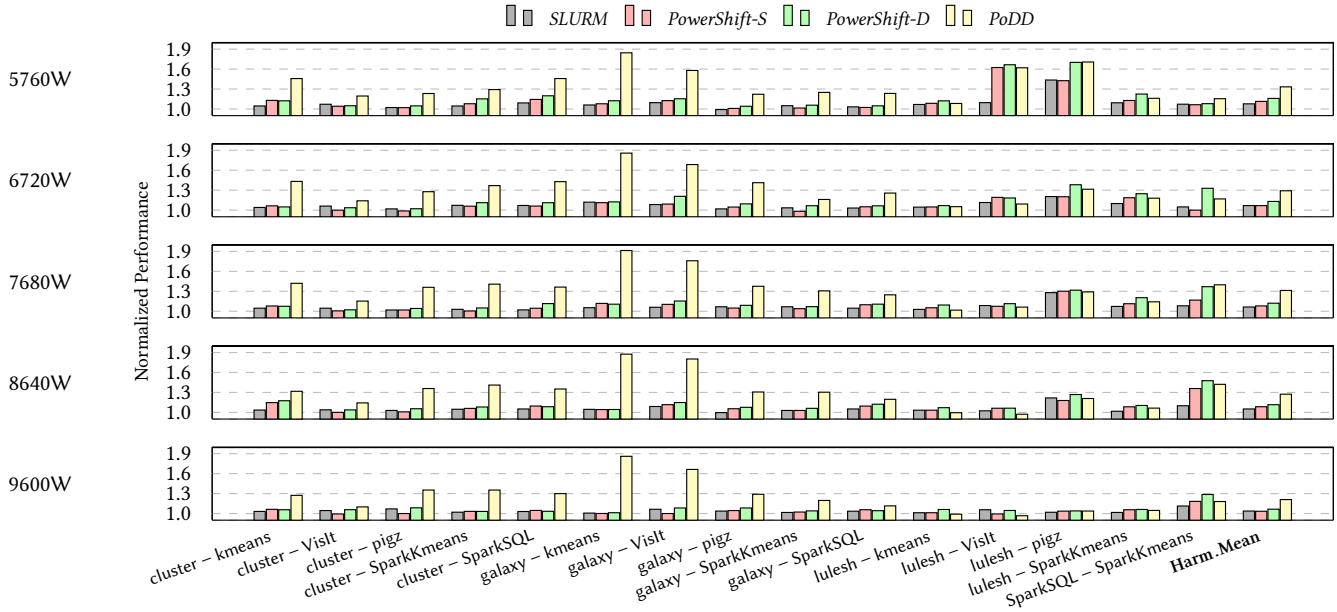

Figure 5: Performance for different power management systems under different power caps.

achieves 6% speedup from dynamically reallocating extra power without awareness of application coupling. *PowerShift-S* achieves 8% speedup from distributing optimal static power between front- and back-end clusters. *PowerShift-D* exploits both of these advantages and achieves 12% speedup on average compared to *Fair*. in comparison, *PoDD* outperforms *Fair* by 28% on average. The large speedup of *PoDD* compared to *PowerShift-D* shows how beneficial it is to coordinate system-level power shifting with advanced node-level power capping.

We note that *PowerShift-D* outperforms *SLURM* because *SLURM* always shifts power from nodes with extra power to nodes running near their power cap. *PowerShift-D*, on the other hand, shifts power from nodes running relatively fast to nodes running relative slowly, allowing the power-hungry nodes running unnecessarily fast to release power to the other nodes. *PoDD* improves over *PowerShift* by incorporating node-level resource classification into the framework so that each node is not only allocated optimal power, but also configured into optimal resource settings.

Fig. 5 shows the performance delivered by each power management system across 5 power budgets for each coupled application pair. All numbers are normalized to that of *Fair*. The 5 charts from top to bottom show results of system power budgets ranging from 5760W to 9600W. The x-axis shows the couple and the y-axis shows the normalized performance. The grouped bars stand for *SLURM*, *PowerShift-S*, *PowerShift-D*, and *PoDD* from left to right. We note three observations from the figure:

(1) Almost all approaches outperform *Fair*, except 2 data points where running `lulesh-visit` with *PoDD*, the performance is slightly worse, but still within 3%.
(2) While a specific application pair might favor one power management approach over another, *PoDD* significantly outperforms *PowerShift* and *SLURM* on average.
(3) For all power management approaches, the performance speedup is generally higher at relatively lower power caps, and lowest at the highest power cap.

Table 5: Resource allocation decision for each application

App	Socket	Hyperthreads	Memory Controller
cluster	single	disabled	dual
lulesh	dual	enabled	either
galaxy	single	disabled	dual
VisIt	dual	enabled	dual
pigz	dual	enabled	single
kmeans	dual	enabled	dual
SparkKmeans	dual	enabled	either
SparkSQL	dual	enabled	either

Here we explain the reasoning for each of these observations. First, *Fair* ensures each node is allocated even power throughout the whole execution. Such power distribution almost always leads to sub-optimal performance for coupled applications as different applications have different power/performance tradeoffs so evenly allocated power results in uneven speeds. For the `lulesh-visit` couple, however, at the two highest power caps both applications already run at maximum speed under *Fair*. Therefore, no power shifting would further speedup either of the coupled applications and *PoDD*'s overhead (and that of other power management approaches) slightly degrades performance.

Second, we consider why some application couples respond better or worse to different power management systems. Table 5 shows the actual resource allocation decision for each application from the classification of phase 1. *We note that for the 8 applications we explore, there are 4 distinct resource configurations found demonstrating that resource assignment is not trivial.* `galaxy` and `cluster` favor a single socket per node without hyperthreading. Because *PoDD* is the only approach that adjusts these node-level resources, it is capable of (1) achieving much higher performance for the nodes running these applications and (2) freeing up additional power from

these nodes to shift to those nodes running the other application in the couple. For example, when `galaxy` is paired with `pigz`, the optimal resource allocation learned from phase 1 allows `galaxy` to use less power with the same performance (because it is best with only a single socket). Then in phase 2, the online model is built based on this optimal configuration, allowing the power freed up from `galaxy` to be shifted to `pigz` for higher couple performance.

`lulesh` and `kmeans` are both compute-intensive benchmarks; when paired with applications offering extra power, their performance greatly improves. However, when they are coupled together, there is little power shifting opportunity because both want more power. `visit` and `pigz` have significant I/O phases, and during those phases they are able to offer extra power to the other application in the couple. Also, `pigz` favors a single memory controller—where most applications favor two—which offers additional configuration optimization opportunities for *PoDD*. At the same time, `pigz` has a dramatic tail effect due to workload imbalance. This imbalance favors the dynamic runtime power shifting. Lastly, the Spark workloads do not have distinct I/O phases, but they do have decent amounts of I/O time scattered across their whole execution time. Even these small I/O times can be utilized by fine-grained dynamic power shifting systems.

Finally, the reason all power capping approaches performs relatively better at lower power caps are twofold. First, the performance of applications tend to scale better with power at lower power cap, as many power/performance tradeoff functions are concave. Second, at higher power caps, the shiftable power is limited. For example, at 200 W per node, as the maximum power cap is 240W, there is only 40W to be shifted around.

Overall, *PoDD* outperforms *Fair* by 28%, *SLURM* by 21%, and *PowerShift* by 14%. These results demonstrate the necessity of (1) optimizing power allocation between coupled applications, (2) dynamically shifting power at runtime, and (3) coordinating advanced node-level power capping with system-wide power shifting.

4.3 Running with Offline Profiles

While offline profiles are not always available, they often exist for a number of applications run repeatedly in large-scale systems. It would be trivial to modify *PoDD* to output its internal power and performance profiles for each application and reload them if the application was run again. In this section, we evaluate *PoDD*'s performance when profiles are provided. Essentially, when offline profiles are provided, *PoDD* would skip phase 1 and phase 2 optimization, and directly enter phase 3 with optimal node-level configuration and power distribution between coupled applications. Table 6 shows the benefit if offline profiles are available. All numbers are still normalized to *Fair*. *PoDD* with profiles gives another 3% performance improvement on average by eliminating the overhead during the convergence of the learning phase and online model building phases. On the other hand, this also reflects how much of overhead learning classifier and online model builder creates.

4.4 Resilience to Tail Effects

Tail effects are a critical issue in distributed workloads, appearing when some small number of nodes has much longer execution time than the majority. These tail nodes drag down overall application performance. In coupled applications, tail nodes slow down both the application to which they belong and the couple.

Table 6: Comparison of *PoDD* with or without profiles.

Power Cap	*PoDD*	*PoDD* with profile
5760W	1.33	1.38
6720W	1.29	1.35
7680W	1.31	1.33
8640W	1.27	1.28
9600W	1.21	1.21
Har. Mean	**1.28**	**1.31**

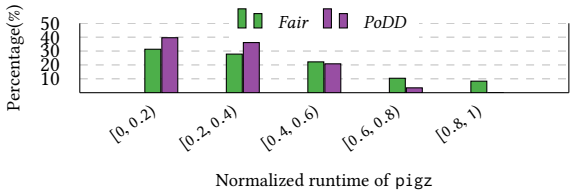

Figure 6: Runtime distribution with work imbalance.

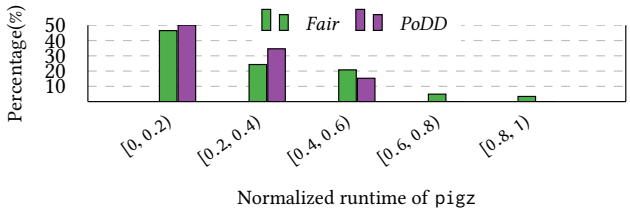

Figure 7: Runtime distribution with system noise.

Table 7: Comparison with different topology mapping.

Power Cap	PowerShift-D	PowerShift-D co-located	*PoDD*	*PoDD* co-located
5760W	1.15	1.23	1.58	1.60
6720W	1.21	1.31	1.68	1.68
7680W	1.15	1.25	1.76	1.70
8640W	1.15	1.24	1.80	1.65
9600W	1.08	1.15	1.66	1.58
Har. Mean	**1.15**	**1.24**	**1.69**	**1.64**

There are various sources that cause tail effects including: system noise and workload imbalance. System noise can come from many forms of physical and human sources and it is inevitable in distributed systems [4]. Workload imbalance has always been one of the great challenges in designing distributed applications. Due to the variations of application workloads and hardware variety, perfect work balance is hard to achieve. Many prior works mitigate workload imbalance in various ways, *PoDD*'s dynamic power management is naturally resilient to these issues.

Fig. 6 shows the runtime distribution of the nodes running the `pigz` application, which is known to have tail effect due to workload imbalance. We have collected the execution time of each node for 3 runs of `pigz` in different coupled pairs under 7680W system-wide power budget. In the histogram, the x-axis is the range of execution time normalized to the longest running node using *Fair*, and the y-axis is the frequency in percentage over all nodes. As we can see, comparing *PoDD* to *Fair*, the numbers in range of $[0.6, 1]$(tail nodes) decrease dramatically from 18.7% to 3.5%. The average coupled performance improvement is 31.3%.

Next, we evaluate the performance under system noise. We emulate system noise by randomly picking two nodes in the `pigz` cluster and run a 24-thread program on each of those nodes stressing the computing resources. All other nodes are left in their usual state.

Table 8: Comparison of online profiling techniques.

Model Name	Accuracy	Convergence Speed	overhead
binary-search	upper bound 2W	4	single core, less than 1s
quadratic	Average 3W	3	single core, less than 2s
logarithm	Average 2W	4	single core, less than 2s

Table 9: Overhead analysis.

Resource	Idle usage	Average usage	Peak usage
CPU	0.10%	0.65%	5.25%
Memory	0.70%	0.75%	1.24%
Network I/O	0.18%	0.20%	0.83%

Table 10: Classification emulation.

Node number	Runtime of serial processing	Runtime of parallel processing
100	17ms	5ms
1,000	159ms	13ms
10,000	1.58s	103ms
100,000	16.6s	0.55s
1,000,000	160s	5s

Table 11: Power pool emulation

Node number	Latency
24	0.7ms
100	1ms
1,000	5ms

This method of generating noise is general, similar (though likely less pronounced) effects could come from other sources such as temperature fluctuations or manufacturing variation. While *PoDD* is not explicitly aware of the inserted system noise, it detects the increased power pressure. Fig. 7 reports the runtime distribution under system noise, all other experimental parameters are the same as those in Fig. 6. Comparing *PoDD* to *Fair*, the numbers in range of $[0.6, 1]$ decrease remarkably from 8.3% to 0%, and the averaged coupled performance speedup is over 44%. *PoDD* achieves these results because it naturally reallocates power from both nodes that finish early and from the other application in the couple to speed up these two tail nodes. This dynamic power management greatly mitigates tail effects in distributed workloads; improving not just the performance of the directly effected but also the overall performance of the coupled pair.

4.5 Topology-obliviousness

PoDD is *topology-oblivious,* meaning the approach can support multiple mappings of applications to physical hardware as long as the power control of the hardware assigned to front- and back-end applications is independent. For simplicity of discussion the majority of evaluation assumes different applications are mapped on physically separate nodes, such separation is not a requirement. This section shows the evaluation when front- and back-end application are scheduled on same nodes but different sockets. Specifically, we evaluate one application couple for which *PoDD* shows great speedup—`galaxy-visit`—across 5 power caps. Table 7 shows the results: that *PoDD* still has great speedup over *PowerShift-D*; however, the relative improvement is less, since the performance of *PowerShift-D* increased in the co-located case. The reason is that in the co-located case, the configurable resources are fewer because there is no option to reduce socket usage. Therefore, *PowerShift-D* happens to be forced to use a better resource allocation than before. Nevertheless, these results show *PoDD* performs well invariant of mapping topology.

4.6 Online Model Builder

In this section, we demonstrate a comparison for 3 different approaches to online model building in terms of their accuracy, convergence speed, and computation overhead. Accuracy is quantified by power error (W) from optimal power allocation. Convergence speed is quantified by how many data points the model needs to converge. Computation overhead is evaluated by how much of computation resources the models need. Table 8 shows the overall comparison. As discussed in Section 3.2, our power range of each end is limited to 35 W, binary search is the best fit in this case, because of high accuracy, low overhead and comparable convergence speed. More complex learning models—*e.g.,* [39, 43, 44]—are not suitable in our case, simply due to comparable or even worse convergence speed with much bigger overhead.

4.7 Scalability Analysis

The controller node learns node-level configurations, builds online power/performance models, and manages the power pool. For our test system, we have 48 computing nodes with one controller. We evaluate the computing, memory, and network I/O stress on the controller node and the time spent exclusively on the controller node (normalized to the entire execution time). Table 9 shows the resource utilization overview. The idle usage, serving as a baseline, is the average usage when machine is nominally idle. As we can see, none of the resources are even close to being challenged supporting a 48-node system. In the following analysis, we run emulations of *PoDD*'s, classifier and power pool at a much larger scale.

Emulation of Classification at Scale: To evaluate the scalability of the light-weight classifier in the learning phase, we emulate the classification with a number of nodes from 100 to 1,000,000. The whole learning process is kept the same as in the real system, except we send many more requests to the learner than we can actually generate on our test system. Table 10 shows the emulation results of both classifying with serial and parallel (using multiple cores within a node) data processing. As we can see, with parallel processing, even at 1M nodes, the runtime overhead from the classifier is only 5 seconds, which is relatively short compared to the long-running applications.

We detail why this scalability is possible. First of all, the classification computation is small and fixed because the input is just a 5-element vector. Second, the time spent in network communication is relatively small as each node will only pass 0.15 KB of data (which could be potentially optimized to only passing 5 floats, if data pre-processing is done locally on each node) to the classifier per application iteration. So, for 1M nodes, that is only 150MB/iteration. Each application iteration lasts from more than a minute to tens of minutes in our case, it comes down to at most 2.5MB/s (if we take the lower bound of 60s as iteration time), which is far from challenging the network capacity of 10Gbps. Lastly, most of the learning time is spent on reading inputs and pre-processing, which

Table 12: Power management system capability comparison.

System	Distributed	Dependent-aware	Dynamic	No offline profiles	No code instrumentation
Fair	X	X	X	✓	✓
PUPiL [62]	X	X	✓	✓	X
SLURM [54]	✓	X	✓	✓	✓
PowerShift-S	✓	✓	X	X	✓
PowerShift-D	✓	✓	✓	X	✓
PoDD	✓	✓	✓	✓	✓

is easily parallelized. In the emulation, we parallelized it with 48 threads on a single multicore server.

Emulation of Power Pool at Scale: We test larger scale systems by emulating multiple virtual nodes per physical node. The virtual nodes request power from the pool as if they were physical nodes. The emulation only scales to 1,000 clients, because beyond that point the physical nodes are overloaded and do not produce reliable request streams. We measure the average latency of receiving a response from the power pool, as shown in Table 11. At 1,000 nodes, the average latency is a negligible 5 ms compared to the period of the control loop of each local decider, which is 1 or 2 seconds. Furthermore, only 16 bytes are exchanged at each request per node, so even at 1M nodes, the upper bound of network bandwidth required is 32MB/s—assuming 1 second control frequency—which again, is far from challenging the network capacity of 10Gbps. Therefore, we expect the power pool structure will scale well.

Based on the bottleneck analysis and emulation results, we predict the controller could handle 1,000 nodes in the worst case and likely scale to tens of thousands nodes with no bottleneck in network or computation.

5 RELATED WORK

Power constraints have become one of the biggest concerns in computer systems at all scales.

At node-level, researchers have proposed both software-based and hardware-based approaches to control power. Early software approaches manage individual components including DVFS for a processor [33], per-core DVFS in a multicore system [28], processor idle-time [21, 64], DRAM [18], and storage [32]. The coordination of multiple system components consistently out-performs approaches that only consider a single component in isolation. Examples include approaches that coordinate processor and DRAM [8, 15, 16, 20, 35, 52] processor speed and core allocation [10, 49], combining DVFS and scheduling [48, 59], memory and disk speed [36] and combining DVFS and process placement [41]. Two recent approaches provide general interfaces to coordinate arbitrary sets of resources to deliver maximum performance for a given power cap [24–26]. These approaches, however, require offline profiles (or models) of power and performance tradeoffs.

For hardware-based approaches, the most widely used and studied is Intel's RAPL—Runtime Average Power Limiting—system, supported in SandyBridge and later processors [11]. Hardware approaches have the advantage of converging to desired power state faster than software approaches. However, they are not able to achieve optimal performance due to only tuning a single resource: DVFS, or processor frequency and voltage [61]. To get the combined benefits of software and hardware, researchers have developed techniques for coordinating the two to achieve high performance with fast convergence [62].

Early work on cluster-level power capping largely ignores performance concerns and focuses on coordinating different levels of the system (data center, rack, server) to ensure power constraints are respected [47]. Given this foundational work to establish system-wide power budgeting, follow up projects could explore improving performance given those budgets. Examples include consolidating workloads to use fewer physical machines [40, 45], job scheduling to achieve resource efficiency [3, 12, 13, 22, 50], and hardware over-provisioning, such that nodes in the system need to be power capped to avoid power loss [51]. Despite differences in methodologies, these systems all try to deal with single application or independent applications and none address the coupled applications studied in this paper.

The closet work to *PoDD*, is *PowerShift* [63]. To the best of our knowledge, *PowerShift* is the first work to propose a power capping solution that specifically addresses the challenges of coupled application workloads. Earlier sections of the paper describe the differences between *PoDD* and *PowerShift* in great detail and demonstrate the performance improvements that *PoDD* obtains.

Table 12 shows the capability comparison of several related power management systems and *PoDD*. Again, *PoDD* offers a practical approach for coupled applications running under system power budget in a large-scale system, delivering high performance efficiency, requiring no prior application profiles, no code instrumentation with decent scalability.

6 CONCLUSION

This paper presents *PoDD*, a hierarchical, distributed, dynamic power management system to address the emerging challenge of power capping coupled workloads in large-scale system. It makes 3 main breakthroughs: (1) developing a novel node-level power capping technique that monitors hardware performance counter and learns the optimal resources allocation using a classifier, (2) coordinating node-level power capping with system-level power shifting for maximized performance, and (3) requiring no application profile by building power/performance models online. Under a variety of power caps running a mixture of different coupled workloads, *PoDD* outperforms several state-of-the-art approaches while showing great resilience to tail effects and system noise. The performance improvements available with *PoDD* demonstrate the benefits of (1) developing power management for coupled workloads (such as future multiphysics and in situ analysis), (2) coordinating node-level power optimization with system-level power shifting, and (3) dynamically shifting power. We hope this work inspires future research and further improvements to these critical problems.

Acknowledgments. We thank the anonymous reviewers for their insightful feedback. This research is primarily supported by a DOE Early Career award. Additional support comes from NSF (CCF-1439156, CCF-1823032, CNS-1764039) and the Proteus project under the DARPA BRASS program. Early prototyping of *PoDD* was performed on the RIVER cluster supported by NSF (CNS-1405959). All results in this paper were collected on the Chameleon testbed supported by NSF. We are extremely grateful to the Chameleon team for making large clusters of bare-metal instances available. Without that resource, we would not have been able to conduct this research.

REFERENCES

[1] Mark Adler. [n. d.]. A parallel implementation of gzip for modern multi-processor, multi-core machines. ([n. d.]). http://zlib.net/pigz/

[2] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Ken Joy, Quincey Koziol, Gerald Lofstead, Jeremy Meredith, Kenneth Moreland, George Ostrouchov, Michael Papka, Venkatram Vishwanath, Matthew Wolf, Nicholas Wright, and Kesheng Wu. 2011. Scientific discovery at exascale: Report from the doe ascr 2011 workshop on exascale data management, analysis,and visualization. (2011).

[3] Peter E Bailey, Aniruddha Marathe, David K Lowenthal, Barry Rountree, and Martin Schulz. 2015. Finding the limits of power-constrained application performance. In *SC*.

[4] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. 2006. Operating System Issues for Petascale Systems. *SIGOPS Oper. Syst. Rev.* 40, 2 (April 2006), 29–33. https://doi.org/10.1145/1131322.1131332

[5] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead. (2008).

[6] Halil Bisgin and HN Dalfes. 2008. Parallel clustering algorithms with application to climatology. In *Geophysical Research Abstracts*, Vol. 10.

[7] J. Chen, Alok Choudhary, S. Feldman, B. Hendrickson, C. R. Johnson, R. Mount, V. Sarkar, V. White, and D. Williams. 2013. Synergistic Challenges in data-intensive science and exascale computing. (2013).

[8] Jian Chen and Lizy Kurian John. 2011. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *ICS*.

[9] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. 2012. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. 357–372.

[10] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *MICRO*.

[11] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: Memory Power Estimation and Capping. In *ISLPED*.

[12] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS*.

[13] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*.

[14] Gökalp Demirci, Ivana Marincic, and Henry Hoffmann. 2018. A Divide and Conquer Algorithm for DAG Scheduling Under Power Constraints. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*.

[15] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *MICRO*.

[16] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. MultiScale: memory system DVFS with multiple memory controllers. In *ISLPED*.

[17] Yi Ding, Nikita Mishra, and Henry Hoffmann. 2019. Generative and Multi-phase Learning for Computer Systems Optimization. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*.

[18] Bruno Diniz, Dorgival Guedes, Wagner Meira, Jr., and Ricardo Bianchini. 2007. Limiting the power consumption of main memory. In *ISCA*.

[19] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *ISCA*.

[20] Wes Felter, Karthick Rajamani, Tom Keller, and Cosmin Rusu. 2005. A performance-conserving approach for reducing peak power consumption in server systems. In *ICS*.

[21] A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy, and J. Kephart. 2009. Power capping via forced idleness. In *Workshop on Energy-Efficient Design*.

[22] Neha Gholkar, Frank Mueller, and Barry Rountree. 2016. Power tuning HPC jobs on power-constrained systems. In *PACT*.

[23] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. 2012. Self-aware computing in the Angstrom processor. In *DAC*.

[24] Henry Hoffmann and Martina Maggio. 2014. PCP: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management. In *ICAC*.

[25] C. Imes and H. Hoffmann. 2016. Bard: A unified framework for managing soft timing and power constraints. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. 31–38. https://doi.org/10.1109/SAMOS.2016.7818328

[26] Connor Imes, Steven Hofmeyr, and Henry Hoffmann. 2018. Energy-efficient Application Resource Scheduling Using Machine Learning Classifiers. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP 2018)*. ACM, New York, NY, USA, Article 45, 11 pages. https://doi.org/10.1145/3225058.3225088

[27] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. 2015. Analyzing and Mitigating the Impact of Manufacturing Variability in Power-constrained Supercomputing. In *SC*.

[28] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *MICRO*.

[29] M. A. Islam, X. Ren, S. Ren, A. Wierman, and X. Wang. 2016. A market approach for handling power emergencies in multi-tenant data center. In *HPCA*. https://doi.org/10.1109/HPCA.2016.7446084

[30] Ian Karlin, Jeff Keasler, and Rob Neely. 2013. *LULESH 2.0 Updates and Changes*. Technical Report LLNL-TR-641973. 1–9 pages.

[31] D. E. Keyes, Lois Curfman McInnes, C. Woodward, W. D. Gropp, E. Myra, and M. Pernice. 2012. Multiphysics Simulations: Challenges and Opportunities. (10/2012 2012). http://hpc.sagepub.com/content/27/1/4.full.pdf+html

[32] Mohammed G. Khatib and Zvonimir Bandić. 2016. PCAP: Performance-aware Power Capping for the Disk Drive in the Cloud. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 227–240. https://www.usenix.org/conference/fast16/technical-sessions/presentation/khatib

[33] C. Lefurgy, X. Wang, and M. Ware. 2008. Power capping: a prelude to power shifting. *Cluster Computing* 11, 2 (2008).

[34] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. 2015. Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 53.

[35] Xiaodong Li, Ritu Gupta, Sarita V. Adve, and Yuanyuan Zhou. 2007. Cross-component energy management: Joint adaptation of processor and memory. *ACM Trans. Archit. Code Optim.* 4, 3 (2007).

[36] Xiaodong Li, Zhenmin Li, Yuanyuan Zhou, and Sarita Adve. 2005. Performance directed energy management for main memory and disks. *Trans. Storage* 1, 3 (2005).

[37] Preeti Malakar, Christopher Knight, Todd Munson, Venkatram Vishwanath, and Michael E. Papka. 2017. Scalable In Situ Analysis of Molecular Dynamics Simulations. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17)*.

[38] Preeti Malakar, Todd Munson, Christopher Knight, Venkatram Vishwanath, and Michael E. Papka. 2018. Topology-aware Space-shared Co-analysis of Large-scale Molecular Dynamics Simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*.

[39] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. 2017. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 31.

[40] Ivana Marincic, Venkatram Vishwanath, and Henry Hoffmann. 2017. PoLiMEr: An Energy Monitoring and Power Limiting Interface for HPC Applications. In *E2SC*.

[41] Andreas Merkel and Frank Bellosa. 2006. Balancing power consumption in multiprocessor systems. In *EuroSys*.

[42] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *ASPLOS*.

[43] N. Mishra, J. D. Lafferty, and H. Hoffmann. 2017. ESP: A Machine Learning Approach to Predicting Application Interference. In *ICAC*.

[44] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. 2015. A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints. In *ASPLOS*.

[45] Ripal Nathuji and Karsten Schwan. 2007. VirtualPower: coordinated power management in virtualized enterprise systems. In *SOSP*.

[46] Nishtala, Rajiv, Marc Gonzalez Tallada, and Xavier Martorell. 2015. A methodology to build models and predict performance-power in CMPS. In *ICPPW*.

[47] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center. *SIGARCH Comput. Archit. News* 36, 1 (March 2008), 48–59. https://doi.org/10.1145/1353534.1346289

[48] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. 2009. Thread motion: fine-grained power management for multi-core systems. In *ISCA*.

[49] S. Reda, R. Cochran, and A.K. Coskun. 2012. Adaptive Power Capping for Servers with Multithreaded Workloads. *Micro, IEEE* 32, 5 (2012).
[50] Haris Ribic and Yu David Liu. 2016. AEQUITAS: Coordinated Energy Management Across Parallel Applications. In *ICS*.
[51] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant V. Kale. 2014. Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget. In *SC*.
[52] Ruchira Sasanka, Christopher J. Hughes, and Sarita V. Adve. 2002. Joint Local and Global Hardware Adaptations for Energy. In *ASPLOS*.
[53] L. Savoie, D. K. Lowenthal, B. R. d. Supinski, T. Islam, K. Mohror, B. Rountree, and M. Schulz. 2016. I/O Aware Power Shifting. In *IPDPS*. https://doi.org/10.1109/IPDPS.2016.15
[54] SLURM. [n. d.]. The SLURM Workload Manager. Online document, https://slurm.schedmd.com/. ([n. d.]).
[55] Volker Springel. 2005. The cosmological simulation code GADGET-2. *Monthly notices of the royal astronomical society* 364, 4 (2005), 1105–1134.
[56] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathost, and Zhiying Wang. 2015. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *MICRO*.
[57] ExaOSR Team. [n. d.]. Key Challenges for Exascale OS/R. Online document, https://collab.mcs.anl.gov/display/exaosr/Challengesl. ([n. d.]).
[58] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. 2010. Conservation cores: reducing the energy of mature computations. In *ASPLOS*.
[59] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*.
[60] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*.
[61] Huazhe Zhang and Henry Hoffmann. 2015. A Quantitative Evaluation of the RAPL Power Control System. In *Proceedings of the 10th International Workshop on Feedback Computing*.
[62] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *ASPLOS*.
[63] Huazhe Zhang and Henry Hoffmann. 2018. Performance and Energy Tradeoffs for Dependent Distributed Applications Under System-wide Power Caps). In *ICPP*.
[64] Xiao Zhang, Rongrong Zhong, Sandhya Dwarkadas, and Kai Shen. 2012. A Flexible Framework for Throttling-Enabled Multicore Management (TEMM). In *ICPP*.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran benchmarks from Parsec-3.0 suite, Rodinia-3.1 suite, NU-Minebench3.0.1 suite, Gadget-2.0.7 suite, Spark-bench, VisIt-2.10.0, pigz-2.3.4, lulesh, parallel-kmeans on 48-node bare metal cluster of Chameleon testbed system. The computing nodes we use are skylake ones with ubuntu16.04 installed. For more library specifications, see our github repo.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:

`https://github.com/podd2019/podd`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: 49-node cluster on chameleon, each server is dual-socket intel Skylake Xeon Gold 6126 CPU, supporting RAPL power capping technology, connected with 32-port software-defined 10 GigE switches.

Operating systems and versions: Ubuntu16.04 running Linux kernel 4.4.0

Compilers and versions: g++ v5.3.1

Applications and versions: Parsec-3.0 suite, Rodinia-3.1 suite, NU-Minebench3.0.1 suite, Gadget-2.0.7 suite, Spark-bench, VisIt-2.10.0, pigz-2.3.4, lulesh, parallel-kmeans

Libraries and versions: cmake-3.0.2, gdal-1.10.0,hdf5-1.8.14, IceT-1-0-0, Imaging-1.1.6, MesaLib-7.10.2, Python-2.7.6, silo-4.10.2, VTK-6.1.0, zlib-1.2.7,

Input datasets and versions: Input data comes with benchmark suite or generated by third-party software, for more detail, refer to the repo

Paper Modifications: cmake-3.0.2, gdal-1.10.0, hdf5-1.8.14, IceT-1-0-0, Imaging-1.1.6, MesaLib-7.10.2, Python-2.7.6, silo-4.10.2, VTK-6.1.0, zlib-1.2.7, netcdf-4.1.1, numpy-1.11.2, pyparsing-1.5.2, requests-2.5.1, setuptools-28.0.0, szip-2.1, Xdmf-2.1.1, gmsh-3.0.5, gsl-2.1, openmpi-2.0.1, fftw-2.1.5,

Output from scripts that gathers execution environment information.

```
SUDO_GID=1011
MAIL=/var/mail/USER
USER=USER
HOME=/home/cc
SUDO_UID=1000
LOGNAME=USER
TERM=xterm-256color
USERNAME=USER
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
↪  in:/sbin:/bin:/snap/bin
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
↪  1;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;0
↪  1:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=3
↪  4;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
↪  .arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*
↪  .lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:
↪  *.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:
↪  *.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=
↪  01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01
↪  ;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;
↪  31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;
↪  31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;
↪  31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;
↪  31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;
↪  35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;
↪  35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;
↪  35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01
↪  ;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=0
↪  1;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=
↪  01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v
↪  =01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv
↪  =01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb
↪  =01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv
↪  =01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=0
↪  1;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=0
↪  1;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=0
↪  0;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=
↪  00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=0
↪  0;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=
↪  00;36:*.xspf=00;36:
SUDO_COMMAND=./collect_environment.sh
SHELL=/bin/bash
SUDO_USER=cc
PWD=/home/cc
+ lsb_release -a
No LSB modules are available.
Distributor ID:        Ubuntu
Description:        Ubuntu 16.04.5 LTS
Release:        16.04
```

```
Codename:        xenial
+ uname -a
Linux powershift-nfs-nfs-server-2wffylhfrcwe
↪  4.4.0-142-generic #168-Ubuntu SMP Wed Jan 16
↪  21:00:45 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6126 CPU
↪  @ 2.60GHz
Stepping:              4
CPU MHz:               1048.328
CPU max MHz:           3700.0000
CPU min MHz:           1000.0000
BogoMIPS:              5189.34
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              19712K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22⌋
↪  ,24,26,28,30,32,34,36,38,40,42,44,46
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23⌋
↪  ,25,27,29,31,33,35,37,39,41,43,45,47
Flags:                 fpu vme de pse tsc msr pae mce
↪  cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪  dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪  pdpe1gb rdtscp lm constant_tsc art arch_perfmon
↪  pebs bts rep_good nopl xtopology nonstop_tsc
↪  aperfmperf pni pclmulqdq dtes64 monitor ds_cpl
↪  vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
↪  dca sse4_1 sse4_2 x2apic movbe popcnt
↪  tsc_deadline_timer aes xsave avx f16c rdrand
↪  lahf_lm abm 3dnowprefetch epb invpcid_single
↪  intel_pt ssbd ibrs ibpb stibp kaiser tpr_shadow
↪  vnmi flexpriority ept vpid fsgsbase tsc_adjust
↪  bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm mpx
↪  avx512f rdseed adx smap clflushopt clwb avx512cd
↪  xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc
↪  cqm_mbm_total cqm_mbm_local dtherm ida arat pln
↪  pts hwp hwp_act_window hwp_epp hwp_pkg_req pku
↪  flush_l1d
+ cat /proc/meminfo
MemTotal:       196554628 kB
MemFree:        192786740 kB
MemAvailable:   194775892 kB
Buffers:           71284 kB
Cached:          2506332 kB
SwapCached:            0 kB
Active:           984516 kB
Inactive:        1689892 kB
Active(anon):      99852 kB
Inactive(anon):    17636 kB
Active(file):     884664 kB
Inactive(file):  1672256 kB
Unevictable:        3652 kB
Mlocked:            3652 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:               312 kB
Writeback:             0 kB
AnonPages:        100680 kB
Mapped:            47256 kB
Shmem:             18172 kB
Slab:             320140 kB
SReclaimable:     141248 kB
SUnreclaim:       178892 kB
KernelStack:       15504 kB
PageTables:         5360 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    98277312 kB
Committed_AS:     462200 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      180000 kB
DirectMap2M:     5658624 kB
DirectMap1G:    196083712 kB
+ inxi -F -c0
System:    Host:
↪  powershift-nfs-nfs-server-2wffylhfrcwe Kernel:
↪  4.4.0-142-generic x86_64 (64 bit) Console: tty 1
           Distro: Ubuntu 16.04 xenial
Machine:   System: Dell product: PowerEdge R740
↪  serial: 7JS20M2
           Mobo: Dell model: 0JM3W2 v: A02 serial:
           ↪  .7JS20M2.CNFCP0079R00CE. Bios: Dell v:
           ↪  1.3.7 date: 02/08/2018
CPU(s):    2 Multi core Intel Xeon Gold 6126s
↪  (-HT-MCP-SMP-) cache: 39424 KB
```

```
           clock speeds: max: 3700 MHz 1: 1325 MHz 2:
           ↪  1036 MHz 3: 1289 MHz 4: 1461 MHz 5:
           ↪  1437 MHz 6: 1469 MHz
           7: 1396 MHz 8: 1486 MHz 9: 1362 MHz 10:
           ↪  1466 MHz 11: 2111 MHz 12: 1416 MHz 13:
           ↪  1306 MHz 14: 1478 MHz
           15: 1492 MHz 16: 1428 MHz 17: 1528 MHz 18:
           ↪  1359 MHz 19: 1177 MHz 20: 1176 MHz 21:
           ↪  1081 MHz
           22: 1256 MHz 23: 1499 MHz 24: 1241 MHz 25:
           ↪  1489 MHz 26: 1013 MHz 27: 1548 MHz 28:
           ↪  1322 MHz
           29: 1565 MHz 30: 1386 MHz 31: 1459 MHz 32:
           ↪  1321 MHz 33: 1299 MHz 34: 1345 MHz 35:
           ↪  1110 MHz
           36: 1256 MHz 37: 1025 MHz 38: 999 MHz 39:
           ↪  1475 MHz 40: 1059 MHz 41: 1186 MHz 42:
           ↪  1362 MHz 43: 1342 MHz
           44: 1292 MHz 45: 1020 MHz 46: 1269 MHz 47:
           ↪  1364 MHz 48: 1251 MHz
Graphics:  Card: Matrox Systems Device 0536
           Display Server: X.org 1.18.4 driver: N/A
           ↪  tty size: 119x40 Advanced Data: N/A
           ↪  for root out of X
Network:   Card-1: Intel I350 Gigabit Network
↪  Connection driver: igb
           IF: eno3 state: down mac: 24:6e:96:62:8e:fa
           Card-2: Intel I350 Gigabit Network
           ↪  Connection driver: igb
           IF: eno4 state: down mac: 24:6e:96:62:8e:fb
           Card-3: Intel Ethernet Controller X710 for
           ↪  10GbE SFP+ driver: i40e
           IF: eno1 state: up speed: 10000 Mbps
           ↪  duplex: full mac: 24:6e:96:62:8e:da
           Card-4: Intel Ethernet Controller X710 for
           ↪  10GbE SFP+ driver: i40e
           IF: eno2 state: down mac: 24:6e:96:62:8e:dc
Drives:    HDD Total Size: 240.1GB (5.9% used) ID-1:
↪  /dev/sda model: MZ7KM240HMHQ0D3 size: 240.1GB
Partition: ID-1: / size: 220G used: 14G (7%) fs: ext4
↪  dev: /dev/sda1
RAID:      No RAID devices: /proc/mdstat, md_mod
↪  kernel module present
Sensors:   None detected - is lm-sensors installed and
↪  configured?
Info:      Processes: 494 Uptime: 27 min Memory:
↪  1175.8/191947.9MB Init: systemd runlevel: 5
          Client: Shell (collect_environ) inxi: 2.2.35
+ lsblk -a
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 223.6G  0 disk
└─sda1   8:1    0 223.6G  0 part /
loop0    7:0    0         0 loop
loop1    7:1    0         0 loop
loop2    7:2    0         0 loop
loop3    7:3    0         0 loop
loop4    7:4    0         0 loop
loop5    7:5    0         0 loop
loop6    7:6    0         0 loop
loop7    7:7    0         0 loop
+ lsscsi -s
[14:0:0:0]   disk    ATA      MZ7KM240HMHQ0D3  GD57
↪  /dev/sda    240GB
[14:0:32:0]  enclosu DP       BP14G+           4.26  -
↪                -
+ module list
./collect_environment.sh: 17:
↪  ./collect_environment.sh: module: not found
+ lshw -short -quiet -sanitize+
cat
H/W path          Device     Class          Description
=================================================┘
↪  ===
                            system         PowerEdge R740
                               ↪  (SKU=NotProvided;Mo┘
                               ↪  delName=PowerEdge
                               ↪  R740)
/0                             bus            0JM3W2
/0/0                           memory         64KiB BIOS
/0/400                         processor      Intel(R)
↪  Xeon(R) Gold 6126 CPU @ 2.60GHz
/0/400/700                     memory         768KiB L1
↪  cache
/0/400/701                     memory         12MiB L2
↪  cache
/0/400/702                     memory         19MiB L3
↪  cache
/0/401                         processor      Intel(R)
↪  Xeon(R) Gold 6126 CPU @ 2.60GHz
/0/401/703                     memory         768KiB L1
↪  cache
/0/401/704                     memory         12MiB L2
↪  cache
/0/401/705                     memory         19MiB L3
↪  cache
/0/1000                        memory         192GiB
↪  System Memory
/0/1000/0                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/1                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/2                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/3                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/4                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/5                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/6                      memory         [empty]
```

```
/0/1000/7                       memory         [empty]
/0/1000/8                       memory         [empty]
/0/1000/9                       memory         [empty]
/0/1000/a                       memory         [empty]
/0/1000/b                       memory         [empty]
/0/1000/c                       memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/d                       memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/e                       memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/f                       memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/10                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/11                      memory         16GiB DIMM
↪  Synchronous 2666 MHz (0.4 ns)
/0/1000/12                      memory         [empty]
/0/1000/13                      memory         [empty]
/0/1000/14                      memory         [empty]
/0/1000/15                      memory         [empty]
/0/1000/16                      memory         [empty]
/0/1000/17                      memory         [empty]
/0/100                          bridge         Intel
↪  Corporation
/0/100/5                        generic        Sky Lake-E
↪  MM/Vt-d Configuration Registers
/0/100/5.2                      generic        Intel
↪  Corporation
/0/100/5.4                      generic        Intel
↪  Corporation
/0/100/8                        generic        Sky Lake-E
↪  Ubox Registers
/0/100/8.1                      generic        Sky Lake-E
↪  Ubox Registers
/0/100/8.2                      generic        Sky Lake-E
↪  Ubox Registers
/0/100/11                       generic        Intel
↪  Corporation
/0/100/11.5                     storage        Lewisburg
↪  SSATA Controller [AHCI mode]
/0/100/14                       bus            Lewisburg
↪  USB 3.0 xHCI Controller
/0/100/14/0       usb2          bus            xHCI Host
↪  Controller
/0/100/14/1       usb1          bus            xHCI Host
↪  Controller
/0/100/14/1/e                   bus            USB hub
/0/100/14/1/e/1                 bus            USB hub
/0/100/14/1/e/4                 bus            USB hub
/0/100/14.2                     generic        Intel
↪  Corporation
/0/100/16                       communication  Lewisburg
↪  CSME: HECI #1
/0/100/16.1                     communication  Lewisburg
↪  CSME: HECI #2
/0/100/16.4                     communication  Lewisburg
↪  CSME: HECI #3
/0/100/17                       storage        Lewisburg
↪  SATA Controller [AHCI mode]
/0/100/1c                       bridge         Lewisburg
↪  PCI Express Root Port #1
/0/100/1c/0       eno3          network        I350
↪  Gigabit Network Connection
/0/100/1c/0.1     eno4          network        I350
↪  Gigabit Network Connection
/0/100/1c.4                     bridge         Lewisburg
↪  PCI Express Root Port #5
/0/100/1c.4/0                   bridge         PLDA
/0/100/1c.4/0/0                 display        Matrox
↪  Electronics Systems Ltd.
/0/100/1f                       bridge         Lewisburg
↪  LPC Controller
/0/100/1f.2                     memory         Memory
↪  controller
/0/100/1f.4                     bus            Lewisburg
↪  SMBus
/0/100/1f.5                     bus            Lewisburg
↪  SPI Controller
/0/2                            bridge         Sky Lake-E
↪  PCI Express Root Port 1C
/0/2/0            eno1          network        Ethernet
↪  Controller X710 for 10GbE SFP+
/0/2/0.1          eno2          network        Ethernet
↪  Controller X710 for 10GbE SFP+
/0/1                            generic        Intel
↪  Corporation
/0/3                            generic        Sky Lake-E
↪  RAS Configuration Registers
/0/4                            generic        Intel
↪  Corporation
/0/6                            generic        Sky Lake-E
↪  CHA Registers
/0/7                            generic        Sky Lake-E
↪  CHA Registers
/0/9                            generic        Sky Lake-E
↪  CHA Registers
/0/a                            generic        Sky Lake-E
↪  CHA Registers
/0/b                            generic        Sky Lake-E
↪  CHA Registers
/0/c                            generic        Sky Lake-E
↪  CHA Registers
/0/d                            generic        Sky Lake-E
↪  CHA Registers
/0/e                            generic        Sky Lake-E
↪  CHA Registers
```

```
/0/f                    generic        Sky Lake-E
↪ CHA Registers
/0/10                   generic        Sky Lake-E
↪ CHA Registers
/0/11                   generic        Sky Lake-E
↪ CHA Registers
/0/12                   generic        Sky Lake-E
↪ CHA Registers
/0/13                   generic        Sky Lake-E
↪ CHA Registers
/0/14                   generic        Sky Lake-E
↪ CHA Registers
/0/15                   generic        Sky Lake-E
↪ CHA Registers
/0/16                   generic        Sky Lake-E
↪ CHA Registers
/0/17                   generic        Sky Lake-E
↪ CHA Registers
/0/18                   generic        Sky Lake-E
↪ CHA Registers
/0/19                   generic        Sky Lake-E
↪ CHA Registers
/0/1a                   generic        Sky Lake-E
↪ CHA Registers
/0/1b                   generic        Sky Lake-E
↪ CHA Registers
/0/1c                   generic        Sky Lake-E
↪ CHA Registers
/0/1d                   generic        Sky Lake-E
↪ CHA Registers
/0/1e                   generic        Sky Lake-E
↪ CHA Registers
/0/1f                   generic        Sky Lake-E
↪ CHA Registers
/0/20                   generic        Sky Lake-E
↪ CHA Registers
/0/21                   generic        Sky Lake-E
↪ CHA Registers
/0/22                   generic        Sky Lake-E
↪ CHA Registers
/0/23                   generic        Sky Lake-E
↪ CHA Registers
/0/24                   generic        Sky Lake-E
↪ CHA Registers
/0/25                   generic        Sky Lake-E
↪ CHA Registers
/0/26                   generic        Sky Lake-E
↪ CHA Registers
/0/27                   generic        Sky Lake-E
↪ CHA Registers
/0/28                   generic        Sky Lake-E
↪ CHA Registers
/0/29                   generic        Sky Lake-E
↪ CHA Registers
/0/2a                   generic        Sky Lake-E
↪ CHA Registers
/0/2b                   generic        Sky Lake-E
↪ CHA Registers
/0/2c                   generic        Sky Lake-E
↪ CHA Registers
/0/2d                   generic        Sky Lake-E
↪ CHA Registers
/0/2e                   generic        Sky Lake-E
↪ CHA Registers
/0/2f                   generic        Sky Lake-E
↪ CHA Registers
/0/30                   generic        Sky Lake-E
↪ CHA Registers
/0/31                   generic        Sky Lake-E
↪ CHA Registers
/0/32                   generic        Sky Lake-E
↪ CHA Registers
/0/33                   generic        Sky Lake-E
↪ CHA Registers
/0/34                   generic        Sky Lake-E
↪ CHA Registers
/0/35                   generic        Sky Lake-E
↪ CHA Registers
/0/36                   generic        Sky Lake-E
↪ CHA Registers
/0/37                   generic        Sky Lake-E
↪ CHA Registers
/0/38                   generic        Sky Lake-E
↪ CHA Registers
/0/39                   generic        Sky Lake-E
↪ CHA Registers
/0/3a                   generic        Sky Lake-E
↪ CHA Registers
/0/3b                   generic        Sky Lake-E
↪ CHA Registers
/0/3c                   generic        Sky Lake-E
↪ CHA Registers
/0/3d                   generic        Sky Lake-E
↪ CHA Registers
/0/3e                   generic        Sky Lake-E
↪ CHA Registers
/0/3f                   generic        Sky Lake-E
↪ CHA Registers
/0/40                   generic        Sky Lake-E
↪ CHA Registers
/0/41                   generic        Sky Lake-E
↪ CHA Registers
/0/42                   generic        Sky Lake-E
↪ CHA Registers
/0/43                   generic        Sky Lake-E
↪ PCU Registers
/0/44                   generic        Sky Lake-E
↪ PCU Registers
```

```
/0/45                   generic        Sky Lake-E
↪ PCU Registers
/0/46                   generic        Sky Lake-E
↪ PCU Registers
/0/47                   generic        Sky Lake-E
↪ PCU Registers
/0/48                   generic        Sky Lake-E
↪ PCU Registers
/0/49                   generic        Sky Lake-E
↪ PCU Registers
/0/4a                   generic        Intel
↪ Corporation
/0/4b                   generic        Sky Lake-E
↪ RAS Configuration Registers
/0/4c                   generic        Intel
↪ Corporation
/0/4d                   generic        Intel
↪ Corporation
/0/4e                   generic        Intel
↪ Corporation
/0/4f                   generic        Intel
↪ Corporation
/0/50                   generic        Intel
↪ Corporation
/0/51                   generic        Intel
↪ Corporation
/0/52                   generic        Intel
↪ Corporation
/0/53                   generic        Intel
↪ Corporation
/0/54                   generic        Intel
↪ Corporation
/0/55                   generic        Intel
↪ Corporation
/0/56                   generic        Intel
↪ Corporation
/0/57                   generic        Intel
↪ Corporation
/0/58                   generic        Intel
↪ Corporation
/0/59                   generic        Intel
↪ Corporation
/0/5a                   generic        Intel
↪ Corporation
/0/5b                   generic        Intel
↪ Corporation
/0/5c                   generic        Intel
↪ Corporation
/0/5d                   generic        Intel
↪ Corporation
/0/5e                   generic        Intel
↪ Corporation
/0/5f                   generic        Intel
↪ Corporation
/0/60                   generic        Intel
↪ Corporation
/0/61                   generic        Intel
↪ Corporation
/0/62                   generic        Intel
↪ Corporation
/0/63                   generic        Intel
↪ Corporation
/0/64                   generic        Intel
↪ Corporation
/0/65                   generic        Intel
↪ Corporation
/0/66                   generic        Intel
↪ Corporation
/0/67                   generic        Intel
↪ Corporation
/0/68                   generic        Sky Lake-E
↪ RAS Configuration Registers
/0/69                   generic        Intel
↪ Corporation
/0/6a                   generic        Intel
↪ Corporation
/0/6b                   generic        Intel
↪ Corporation
/0/6c                   generic        Intel
↪ Corporation
/0/6d                   generic        Intel
↪ Corporation
/0/6e                   generic        Intel
↪ Corporation
/0/6f                   generic        Intel
↪ Corporation
/0/70                   generic        Sky Lake-E
↪ M3KTI Registers
/0/71                   generic        Sky Lake-E
↪ M3KTI Registers
/0/72                   generic        Sky Lake-E
↪ M3KTI Registers
/0/73                   generic        Sky Lake-E
↪ M3KTI Registers
/0/74                   generic        Sky Lake-E
↪ M3KTI Registers
/0/75                   generic        Sky Lake-E
↪ M2PCI Registers
/0/76                   generic        Sky Lake-E
↪ M2PCI Registers
/0/77                   generic        Sky Lake-E
↪ M2PCI Registers
/0/78                   generic        Sky Lake-E
↪ M2PCI Registers
/0/79                   generic        Sky Lake-E
↪ MM/Vt-d Configuration Registers
/0/7a                   generic        Intel
↪ Corporation
```

```
/0/7b                       generic       Intel
↪ Corporation
/0/7c                       generic       Sky Lake-E
↪ Ubox Registers
/0/7d                       generic       Sky Lake-E
↪ Ubox Registers
/0/7e                       generic       Sky Lake-E
↪ Ubox Registers
/0/101                      bridge        Sky Lake-E
↪ PCI Express Root Port 1A
/0/101/0          scsi14    storage       MegaRAID
↪ SAS-3 3008 [Fury]
/0/101/0/0.0.0    /dev/sda  disk          240GB
↪ MZ7KM240HMHQ0D3
/0/101/0/0.0.0/1  /dev/sda1 volume        223GiB
↪ EXT4 volume
/0/101/0/0.20.0             generic       BP14G+
/0/7f                       generic       Intel
↪ Corporation
/0/80                       generic       Sky Lake-E
↪ RAS Configuration Registers
/0/81                       generic       Intel
↪ Corporation
/0/82                       generic       Sky Lake-E
↪ CHA Registers
/0/8.1                      generic       Sky Lake-E
↪ CHA Registers
/0/8.2                      generic       Sky Lake-E
↪ CHA Registers
/0/83                       generic       Sky Lake-E
↪ CHA Registers
/0/84                       generic       Sky Lake-E
↪ CHA Registers
/0/85                       generic       Sky Lake-E
↪ CHA Registers
/0/86                       generic       Sky Lake-E
↪ CHA Registers
/0/87                       generic       Sky Lake-E
↪ CHA Registers
/0/88                       generic       Sky Lake-E
↪ CHA Registers
/0/89                       generic       Sky Lake-E
↪ CHA Registers
/0/8a                       generic       Sky Lake-E
↪ CHA Registers
/0/8b                       generic       Sky Lake-E
↪ CHA Registers
/0/8c                       generic       Sky Lake-E
↪ CHA Registers
/0/8d                       generic       Sky Lake-E
↪ CHA Registers
/0/8e                       generic       Sky Lake-E
↪ CHA Registers
/0/8f                       generic       Sky Lake-E
↪ CHA Registers
/0/90                       generic       Sky Lake-E
↪ CHA Registers
/0/91                       generic       Sky Lake-E
↪ CHA Registers
/0/92                       generic       Sky Lake-E
↪ CHA Registers
/0/93                       generic       Sky Lake-E
↪ CHA Registers
/0/94                       generic       Sky Lake-E
↪ CHA Registers
/0/95                       generic       Sky Lake-E
↪ CHA Registers
/0/96                       generic       Sky Lake-E
↪ CHA Registers
/0/97                       generic       Sky Lake-E
↪ CHA Registers
/0/98                       generic       Sky Lake-E
↪ CHA Registers
/0/99                       generic       Sky Lake-E
↪ CHA Registers
/0/9a                       generic       Sky Lake-E
↪ CHA Registers
/0/9b                       generic       Sky Lake-E
↪ CHA Registers
/0/9c                       generic       Sky Lake-E
↪ CHA Registers
/0/9d                       generic       Sky Lake-E
↪ CHA Registers
/0/9e                       generic       Sky Lake-E
↪ CHA Registers
/0/9f                       generic       Sky Lake-E
↪ CHA Registers
/0/a0                       generic       Sky Lake-E
↪ CHA Registers
/0/a1                       generic       Sky Lake-E
↪ CHA Registers
/0/a2                       generic       Sky Lake-E
↪ CHA Registers
/0/a3                       generic       Sky Lake-E
↪ CHA Registers
/0/a4                       generic       Sky Lake-E
↪ CHA Registers
/0/a5                       generic       Sky Lake-E
↪ CHA Registers
/0/a6                       generic       Sky Lake-E
↪ CHA Registers
/0/a7                       generic       Sky Lake-E
↪ CHA Registers
/0/a8                       generic       Sky Lake-E
↪ CHA Registers
/0/a9                       generic       Sky Lake-E
↪ CHA Registers
```

```
/0/aa                   generic        Sky Lake-E
↪  CHA Registers
/0/ab                   generic        Sky Lake-E
↪  CHA Registers
/0/ac                   generic        Sky Lake-E
↪  CHA Registers
/0/ad                   generic        Sky Lake-E
↪  CHA Registers
/0/ae                   generic        Sky Lake-E
↪  CHA Registers
/0/af                   generic        Sky Lake-E
↪  CHA Registers
/0/b0                   generic        Sky Lake-E
↪  CHA Registers
/0/b1                   generic        Sky Lake-E
↪  CHA Registers
/0/b2                   generic        Sky Lake-E
↪  CHA Registers
/0/b3                   generic        Sky Lake-E
↪  CHA Registers
/0/b4                   generic        Sky Lake-E
↪  CHA Registers
/0/b5                   generic        Sky Lake-E
↪  CHA Registers
/0/b6                   generic        Sky Lake-E
↪  CHA Registers
/0/b7                   generic        Sky Lake-E
↪  CHA Registers
/0/b8                   generic        Sky Lake-E
↪  CHA Registers
/0/b9                   generic        Sky Lake-E
↪  CHA Registers
/0/ba                   generic        Sky Lake-E
↪  CHA Registers
/0/bb                   generic        Sky Lake-E
↪  CHA Registers
/0/bc                   generic        Sky Lake-E
↪  PCU Registers
/0/bd                   generic        Sky Lake-E
↪  PCU Registers
/0/be                   generic        Sky Lake-E
↪  PCU Registers
/0/bf                   generic        Sky Lake-E
↪  PCU Registers
/0/c0                   generic        Sky Lake-E
↪  PCU Registers
/0/c1                   generic        Sky Lake-E
↪  PCU Registers
/0/c2                   generic        Sky Lake-E
↪  PCU Registers
/0/c3                   generic        Intel
↪  Corporation
/0/c4                   generic        Sky Lake-E
↪  RAS Configuration Registers
/0/c5                   generic        Intel
↪  Corporation
/0/8                    generic        Intel
↪  Corporation
/0/c6                   generic        Intel
↪  Corporation
/0/c7                   generic        Intel
↪  Corporation
/0/c8                   generic        Intel
↪  Corporation
/0/c9                   generic        Intel
↪  Corporation
/0/ca                   generic        Intel
↪  Corporation
/0/cb                   generic        Intel
↪  Corporation
/0/cc                   generic        Intel
↪  Corporation
/0/cd                   generic        Intel
↪  Corporation
/0/ce                   generic        Intel
↪  Corporation
/0/cf                   generic        Intel
↪  Corporation
/0/d0                   generic        Intel
↪  Corporation
/0/d1                   generic        Intel
↪  Corporation
/0/d2                   generic        Intel
↪  Corporation
/0/d3                   generic        Intel
↪  Corporation
/0/d4                   generic        Intel
↪  Corporation
/0/d5                   generic        Intel
↪  Corporation
/0/d6                   generic        Intel
↪  Corporation
/0/d7                   generic        Intel
↪  Corporation
/0/d8                   generic        Intel
↪  Corporation
/0/d9                   generic        Intel
↪  Corporation
/0/da                   generic        Intel
↪  Corporation
/0/db                   generic        Intel
↪  Corporation
/0/dc                   generic        Intel
↪  Corporation
/0/dd                   generic        Intel
↪  Corporation
/0/de                   generic        Intel
↪  Corporation
```

```
/0/5                        generic        Intel
↪ Corporation
/0/5.2                      generic        Sky Lake-E
↪ RAS Configuration Registers
/0/5.4                      generic        Intel
↪ Corporation
/0/df                       generic        Intel
↪ Corporation
/0/e0                       generic        Intel
↪ Corporation
/0/e1                       generic        Intel
↪ Corporation
/0/e2                       generic        Intel
↪ Corporation
/0/e3                       generic        Intel
↪ Corporation
/0/e4                       generic        Intel
↪ Corporation
/0/e5                       generic        Sky Lake-E
↪ M3KTI Registers
/0/e6                       generic        Sky Lake-E
↪ M3KTI Registers
/0/e7                       generic        Sky Lake-E
↪ M3KTI Registers
/0/e8                       generic        Sky Lake-E
↪ M3KTI Registers
/0/e9                       generic        Sky Lake-E
↪ M3KTI Registers
/0/ea                       generic        Sky Lake-E
↪ M2PCI Registers
/0/eb                       generic        Sky Lake-E
↪ M2PCI Registers
/0/ec                       generic        Sky Lake-E
↪ M2PCI Registers
/0/ed                       generic        Sky Lake-E
↪ M2PCI Registers
/1                          power          0CMPGMA00
/2                          power          0CMPGMA00
```