A Unified Platform for Exploring Power Management Strategies

Daniel Ellsworth*†, Tapasya Patki†, Martin Schulz†, Barry Rountree†, Allen D. Malony*
*University of Oregon, Eugene, Oregon, USA
†Lawrence Livermore National Laboratory Livermore, California, USA
{dellswor,malony}@cs.uoregon.edu, {patki1,schulzm,rountree4}@llnl.gov

***Abstract*—Power is quickly becoming a first class resource management concern in HPC. Upcoming HPC systems will likely be hardware over-provisioned, which will require enhanced power management subsystems to prevent service interruption. To advance the state of the art in HPC power management research, we are implementing SLURM plugins to explore a range of power-aware scheduling strategies. Our goal is to develop a coherent platform that allows for a direct comparison of various power-aware approaches on research as well as production clusters.**

I. Introduction

HPC systems deployed by the United States Department of Energy (DOE) are currently capable of just under 20 petaFLOPS[1] while using about 10 megawatts of power [1]. The DOE has articulated a goal for delivering a first generation exascale system[2] within a 20 megawatt power budget [2], [3]. Significant advances in energy efficiency will be needed in both hardware and software in order to meet this goal.

HPC systems are typically operated in the interests of advancing underlying scientific goals, which makes it difficult to measure their performance quantitatively. Common metrics for such performance comparison include *job throughput* (or, the number of jobs completed per unit time), *node utilization* (or, the percentage of total nodes in the cluster allocated to executing jobs over a period of time), and FLOPS.

Energy efficiency relates to the energy cost to complete a computation; energy efficient solutions use less energy to complete the same computation. Improving the energy efficiency of applications often results in variation of power consumption over time because of the instruction mix and associated data movements. In energy-constrained environments, such as mobile computing, the aforementioned power variability is a good trade-off for extending battery life and reducing costs. Many of the innovations from the mobile space are directly applicable to the energy efficiency challenges faced by the HPC community.

Future HPC systems, however, are expected to be *power-constrained* instead of being energy-constrained. In the past, HPC systems have provisioned sufficient power to be able to operate all components at their theoretical peak consumption. Power grid infrastructure, electricity provider, and operational cost limitations place a hard limit on the rate at which electricity can be delivered to a system. As a result, HPC systems now have to operate under a finite power envelope and realize no direct benefit for utilizing less than the provisioned power.

Hardware over-provisioning is a physical HPC system design that allows additional work to be performed within the same power envelope. In an over-provisioned system, more computing hardware is available than can be powered simultaneously at peak consumption. Power savings in one part of the system can be redirected to the *extra* hardware and used there for computation. The availability of additional hardware supports executing more applications as well as improving the performance of some applications by scaling them out, thus improving throughput and utilization. Based on published consumption traces from three high-end DOE HPC systems, namely Vulcan, Sequoia, and Titan [4], [5], opportunities exist to utilize roughly 25% more hardware and remain safely within the power budget most of the time, while the remaining short times are peaks that can be controlled by actively enforcing power caps.

For hardware over-provisioned systems to be practical, enforcement of such system wide power caps must occur with sufficient temporal resolution to avoid damaging overloads to the power distribution infrastructure. Technologies such as Intel's running average power limit (RAPL), which provide a hardware mechanism that can enforce component level power caps with fine temporal resolution, will need to be available.

Another practical barrier to the deployment of hardware over-provisioned systems is resolving how power should be scheduled across the cluster. Recently, it has been suggested that power should be treated as a schedulable resource. Estimating the power and performance of HPC applications can thus be used for efficient scheduling. While simple static approaches that set a uniform power cap across nodes in a job are safe for overprovisioned systems, they often result in limited throughput. This is because HPC applications can exhibit distinct runtime behavior resulting in phases with low and high power consumption. Thus, runtime systems with the capability to dynamically modify power allocations while jobs are executing have also been proposed. More research is needed to directly compare and better understand the trade-off space for power scheduling strategies. A general platform for conducting such research is currently missing.

[1]FLOPS: floating point operations per seconds
[2]System capable of 10^{18} FLOPS, or 1000 petaFLOPS.

E2SC2016; Salt Lake City, Utah, USA; November 2016
978-1-5090-3856-5/16/$31.00

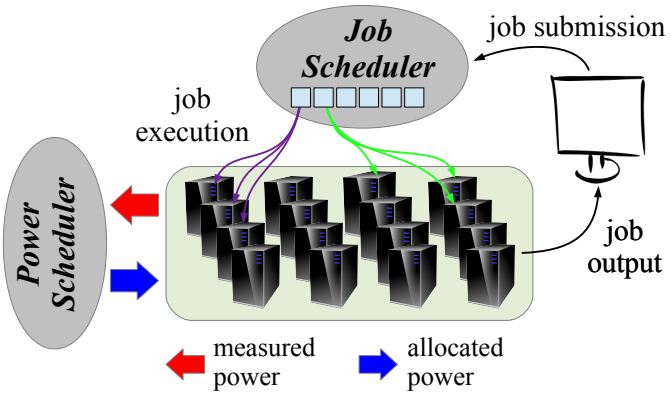

Fig. 1. Model of a hardware over-provisioned system with the job and power schedulers.

In this paper, we present the design and implementation of such a platform for exploration of power scheduling strategies. Our platform is based on SLURM, and, while not production ready, is suitable for large-scale experiments on research clusters. Using a common platform to analyze the power-aware scheduling space allows for direct comparison of different approaches using the same metrics. Section II presents background and related work. Sections III to V discuss the design concerns, evaluation parameters and implementation details of our SLURM extensions. We conclude in Section VI.

II. Background

We present a high-level system model for a hardware over-provisioned HPC system in Figure 1, highlighting the *power scheduler* and the *job scheduler*. The power and job scheduling roles are separated in the model in order to simplify reasoning about whether the system is guaranteed to adhere to a power bound and to simplify the verification of correctness of the power management approach. The power scheduler's primary objective is to maintain the system power bound, with all other objectives being secondary. The job scheduler's primary objective is to efficiently execute work in an order that aligns with organizational objectives, which is a complex multi-objective optimization problem. While the job and power schedulers should coordinate to efficiently execute work within the system power bound, the power scheduler must always be empowered to arbitrarily degrade performance if needed to protect the system.

The current state of the practice for over-provisioned systems is to statically allocate the same amount of power to every node within the cluster and never change the allocation. Unfortunately, static power caps often result in poor power utilization and throughput. Better power management strategies involve *application awareness*, where power allocations are set based on an application's anticipated power consumption, and *dynamic power redistribution*, where power allocations are shifted between components.

SLURM [6] is production resource manager that is currently distributed with two power management systems. On platforms supporting the correct model specific registers (MSRs), a dynamic voltage and frequency scaling (DVFS) based solution is provided [7]. The DVFS solution does not redistribute power and cannot guarantee safe operation. On Cray systems, a plugin can be configured to dynamically redistribute power [8] using RAPL. Use of the Cray plugin is limited to Cray systems and has a low temporal resolution.

Common themes for power scheduling research include energy efficiency, performance optimization, and throughput optimization. Energy efficiency studies analyze applications and often involve slowing non-critical path computations via DVFS. Research on performance optimization strives to achieve the least execution time for an application given a power cap or an energy saving goal [9], [10]. Throughput optimization looks at minimizing the average turnaround time of jobs [4], [11]. Our SLURM extensions support studies focusing job throughput optimizations.

Sarood [11] applies an integer linear programming approach to schedule power and moldable jobs within a data center. This strategy (PARM) uses DVFS and has a goal of maximizing throughput. Its scalability, however, limited: authors report that 15 seconds were required for making decisions on a queue with 200 jobs.

Work by Patki, et al. [4] (RMAP) assumes jobs are moldable and that a model for estimating application performance is available. It introduces *power-aware backfilling* and is implemented in a SLURM simulator. The best proposed policy in RMAP improves power utilization by 17% and system throughput by 32%.

Ellsworth, et al. [12] present PowSched, which uses the difference between power consumption measurements and power allocation to shift power across an HPC cluster without awareness of jobs. Emulation experiments indicate very good scaling and experimental results have shown a significant improvement over naive strategies when high and low consumption phases are favorably aligned.

Savoie, et al. [13] make a distinction between shifting and scheduling algorithms. Shifting algorithms redistribute a resource across concurrently active work; an activity this paper will refer to as power scheduling. Scheduling algorithms control when work starts; an activity this paper will refer to as job scheduling.

Cao, et al. [14] present a demand aware power scheduler with some job scheduler integration. Dynamic power balancing and the ability to launch jobs that would otherwise be denied can increase job throughput.

Hierarchical power scheduling has been recently suggested to address future scaling challenges. Ellsworth, et al. [15] describe the hierarchical power scheduling approach in the Argo ExaOS/R project. Gholkar, et al. [16] report on a two-level power scheduler that addresses processor performance variations. PPartition allocates power across jobs and PTune allocates power across nodes within a job.

III. Design Concerns

In this section, we highlight the need for a common evaluation platform and the considerations that led to our extension of a production scheduler.

A. Comparability

In general, comparison between published scheduling techniques is difficult due to differences in the evaluation environments. Consider some of the power schedulers from the related work, PARM [11], RMAP [4], PowSched [12], and PPartition [16]. Each of them uses different workloads, has different job queue capabilities, compares against different baseline power strategies, and are evaluated on different platforms with different node counts. Ellsworth, et al. [12] show that performance is strongly impacted by workload mix; the lack of a uniform workload makes reported improvement numbers non-comparable. PPartition and RMAP both leverage backfilling by the scheduler to get performance gains but use different backfilling techniques. In contrast, PowSched is experimentally evaluated with only static job schedules. It is not clear if PARM's baseline includes power cap maintenance, which is explicitly in the baseline for RMAP, PowSched, and PPartition. Published research work is thus insufficient to fairly evaluate solutions for production systems.

Comparison of published solutions via installation on a single testbed is also challenging. RMAP was only evaluated in simulation and code for experimental evaluation on real hardware is not available. Since each solution has a different job scheduler, there would be no uniform way to package and launch equivalent workloads for each solution. Even if the workloads could be described uniformly for launch, the default job scheduler features differ significantly between the solutions. Comparing a statically scheduled queue from PowSched with a backfilling scheduler, like PParition or RMAP, is an invalid comparison. Solving the workload description and queue features problems would still leave measurement collection as an open issue. Implementing several power management strategies within a single platform should avoid these complexities by standardizing the inputs, outputs, and overheads used across compared techniques.

B. Production Support

Simulation studies are useful to get a sense of a system's anticipated behavior. However, simulation cannot take into account the complex low-level interactions between components or quantify interface overheads. Under the best of circumstances, distributed applications suffer from variability due to minor time differences and shared subsystems, such as the network. The introduction of component level power capping increases variability further [17]. The inability of models to fully capture the complex interrelated behavior make empirical experiments necessary.

Implementation of the power management strategies on a research scheduler is tempting. Working within a simple scheduler codebase should reduce development time and support better focus on algorithmic details. However, such research codes exhibit missing feature richness and barriers to adoption, including the need for re-implementation in order to move to production. Minimizing the distance between research and practice is an important practical objective for our work.

An additional benefit of building on top of a production scheduler codebase is the ability to leverage the depth of existing scheduling work. Reimplementation of features such as backfilling, resource accounting, and priority policies would require significant effort. Implementing reliability and scaling features would also involve significant engineering effort outside of the power management research focus.

IV. Evaluation Parameters

In this section, we highlight some of parameter space we plan to explore and evaluate using our platform.

A. Decision Time

Understanding when power allocation decisions are made provides a coarse grain parameter for classifying and comparing different approaches. Three granularities of time are of primary interest for binning approaches: machine lifetime, job lifetime, and arbitrary intervals.

Machine lifetime Node power allocations are set just once when the machine is deployed and never changed. Existing systems effectively use this strategy since all components may safely consume up to the thermal design power (TDP).

Job lifetime At the time a job is scheduled, the power allocation for the participating nodes is set and not changed while the job runs. Research systems like RMAP use this strategy.

Arbitrary intervals Power allocations may be changed on any node at any time. Research systems like PowSched use this strategy.

While machine lifetime approaches are inherently static and have been shown to underperform[12], approaches based on job lifetime and arbitrary intervals are interesting to study. Power scheduling at job boundaries is a natural fit for existing batch resource scheduling techniques. Arbitrary intervals move power scheduling in the direction of runtime system research. Adjusting power during an application's execution allows phase behavior to be leveraged to improve performance.

B. Fairness

A common concern for the resource scheduling community is understanding the fairness guarantees provided by a scheduling algorithm. Our platform should permit different notions of fairness to be explored. Two possible fairness considerations we plan to explore are power fairness and node fairness.

Power fairness A job requesting resources receives a fair allocation of power based on the node count of the resource request. Optimizations and savings occur by adjusting other attributes, such as job node count and work placement.

Node fairness A job requesting node resources receives the requested node count and placement, but may suffer performance degradations due to systemwide power bounds.

Fundamentally, hardware over-provisioning relies on unfair power allocations across nodes to improve throughput. To provide power fairness, the node count must mutable. Similarly, providing node fairness requires the power allocation be mutable.

While understanding scheduler behavior in terms of fairness can be informative, deploying organizations are much more likely to care about work priority and throughput. Fairness in execution order is unlikely to be the primary objective. For example, highly desirable scheduling optimizations such as backfilling can increase system utilization but erode execution order fairness. Increasing utilization and job throughput is likely realized only by taking advantage of imbalances rather than enforcing fairness.

C. Job Awareness

A major research interest motivating the platform is understanding how different levels of job awareness impact the quality of the generated allocations. Our existing work, RMAP [4] and PowSched [12], can be seen as existing on opposite ends of a spectrum of job awareness. For RMAP, most of the job properties are known and assumed to be well modeled before jobs are executed, requiring substantial resources for model generation. The detailed a priori knowledge allows RMAP to achieve improved turnaround time by changing a job's configuration at launch time based on power and other resource constraints. PowSched, on the other hand, achieves good performance to evict all concurrent work with no awareness of jobs, nodes, or time beyond the most recent scheduling interval. We hypothesize that a more job aware PowSched could make better decisions and an RMAP could still make good decisions with weaker models.

Also related to job awareness is the level and direction of scheduler communication. Recall from the system model (Figure 1) that the job and power schedulers are logically separated to aid in verification. If job awareness is useful for power scheduling, a power scheduler should receive job information from the job scheduler. Some exploration of what job information is useful and how the information can be used to produce better power schedules is needed. Job schedulers should also receive power information from the power scheduler to get feedback on the power utilization of the existing schedule for use in generating later job schedules. Of particular interest is how intervals of power oversubscription might be handled when power and job schedulers coordinate action.

V. Architecture

Due to its wide spread popularity and our prior RMAP work, we selected SLURM as the base production job scheduler to build on. Its plugin infrastructure was a good fit for our needs. Much of the resource scheduling work is handled in SLURM with the help of node *select plugins* that can be changed through configuration files. A *power plugin* interface is also available in the codebase. However, the only implemented power plugin is limited to Cray systems.

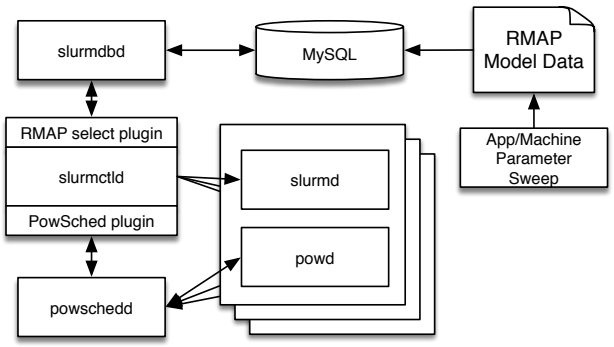

Fig. 2. Design of our proposed SLURM extensions for RMAP and PowSched

A. High-level Architecture

The core architectural design of SLURM has remained largely intact over more than a decade of development [6]. The centralized control daemon, `slurmctld`, communicates with a daemon on each node, `slurmd`. `Slurmctld` directs `slurmd` instances to start and stop work on nodes, and the `slurmd` instances report node configuration and status. Plugins are used to tailor SLURM for different environments.

SLURM user utilities, such as `srun` and `squeue`, interact with `slurmctld` to change and interrogate the job and resource queues. A database daemon, `slurmdbd`, is included to securely manage accounting information stored in a relational database.

Our design goals can be met primarily through the implementation of two plugins. A *Select plugin* determines which nodes and node resources are to be given to a job. RMAP can be implemented almost completely as a *Select plugin*. Additional minor extensions are needed to `slurmdbd`, `srun`, and SLURM's RPC protocol to support the power model information required for RMAP. A *power plugin* allows power bounds to be set on nodes participating in a job. The power plugin interface can be used to generate a bridge between SLURM and an external power scheduler like PowSched. Some changes are needed to the existing power plugin API to make the API more generic. Figure 2 presents this design.

Existing data structures within SLURM are used to communicate between the *select* and *power* plugins. During normal operation, SLURM *select* and *power* plugins receive node records and job records. These records are implemented as `structs` and include pointers to opaque plugin specific structures. Hints regarding node power constraints, as determined by the *power* plugin, can be included in the node records received by the *select* plugin. Hints regarding a job's power needs, as determined by the *select* plugin, can be included in the job record received by the *power* plugin. These plugins are discussed in detail below.

B. RMAP as a Select Plugin

Select plugins in SLURM, such as `cons_res`, `linear`, and `bluegene`, are responsible for selecting the specific resources a job should use at runtime and to indicate if sufficient resources exist to launch a candidate job. When determining if a job can be scheduled, SLURM provides the *Select* plugin with a job description via a `job record struct pointer`, and the available resources via a `bitmask pointer`. If the available resources are insufficient to launch the job, the *Select* plugin returns an error code. The *Select* plugin will update the bitmask to indicate the selected nodes and the job record will be updated with fine grain resource assignments (for example, specific cores).

RMAP achieves performance improvements by optimizing the job configuration and molding the job to the available resources. Using an application specific model, RMAP selects the most efficient resource configurations for a job. When the available resources are insufficient for the most efficient configuration, RMAP will consider scheduling the job with slightly reduced resources. If RMAP determines that the performance degradation from reduced resources is not within user-specified acceptable limits, it will inform the scheduler that the job can not be run and should be requeued. The *Select* plugin is architecturally good location for RMAP implementation.

RMAP was originally implemented as a *Select* plugin on the BCS SLURM simulator [4], but the simulation implementation could not be moved directly over to production. Simulation RMAP used `Job ID` for application-specific model lookups with the help of an internal mapping, which is clearly unreasonable for a live system. It also directly accessed a model database that assumed that the SLURM database is co-resident with the `slurmctld`, which violates the SLURM security model. SLURM's RPC protocol and `slurmdbd` have now been extended to support the RMAP model lookups. The data structures handled by the *Select* plugin have been augmented to provide suggested power configurations to the power plugin.

1) Job Model Flag: Power consumption and performance properties of a job depend on the application, resource configuration, and data. Rather than attempt to analyze the job record for hints regarding the correct model, the RMAP implementation expects the user or another interfacing software system to identify the correct model at job submission time. A switch has been added to `srun`, *–model*, to pass the job model identifier. `srun` packs the identifier into the job record sent to the `slurmctld`. If no job model is given, the *RMAP Select* plugin does not attempt to mold the job when selecting resources. When a job model is given, the *RMAP Select* plugin queries the model database, via slurmdbd, and configures the job based on the returned resource configuration.

For implementation, the option parser was extended to be aware of the new flag, requiring changes to `libsrun`. The *Select* plugin API allows plugin specific key value pairs to be added to the job record. This feature is used by our modified `libsrun` to include the model identifier with the rest of the job attributes. Our `srun` modifications are safe to use with *non-RMAP select* plugins since the underlying SLURM data structures have not been altered.

2) Model Database: An RMAP job model is given as a set of tuples. The tuples are specific to the machine, application, and data used for the job. Each tuple contains the execution time, power, node count, and core count associated with a job. To determine if a job can be run, the *RMAP select* plugin must query the job model for a tuple that fits within the available power and node count constraints. If a job can be run, the plugin should mold the selected job resources to match the values returned by the model. The query requirements make a relational database a good fit for model storage and SLURM has existing support for MySQL.

At many sites, MySQL is used by SLURM to store accounting information. `Slurmdbd` provides controlled access to the database and prevents tampering. Components other than the `slurmdbd` load the *slurmdbd accounting storage* plugin, which then handles the RPC calls with the `slurmdbd` daemon.

Adding access to the RMAP model was the most invasive of our extensions. The actual database logic was placed in the *mysql accounting storage* plugin since most other database accesses occur here. Accounting storage extension required adding a new call to the accounting storage plugin API and implementing functions in the *accounting storage* plugin. RPC extension involved adding additional message types, structs, data marshaling and unmarshaling logic, and some minor message handling code.

3) Resource Molding: Job molding follows directly from the configuration returned by the model query. The *select* plugin includes the `Model ID` and resource constraints when making the RPC call to the model database. Return values from the RPC call include the detailed resource configuration for the job. The *select* plugin ignores the requested resources in the job record and selects job resources to match the configuration returned from the model query.

4) Model Generation: To select efficient application configurations, detailed models of application performance are required. These are generated by first gathering application profile data and then developing linear models for performance prediction. Profiling the application includes gathering execution time and maximum power consumption at a few selected node and core counts while varying node-level power caps and turbo boost options. Application-specific linear models to predict both execution time and overall power consumption can then be developed based on this data. Such models have been shown to have reasonable accuracy, with median errors in the range of 5-10% [4]. Developing a general model to predict performance for a set of applications with varying parameters is an orthogonal research problem.

C. PowSched as a Power Plugin

The *power* plugin is primarily an interface between the power and the job scheduler. Using the *power* plugin as a gateway between separate systems echoes SLURMs early

approach to separating job and resource scheduling. Our initial *power* plugin only relays job information to the power scheduler. However, later iterations will allow the power scheduler to request behavior from the resource manager. A functionality of particular interest is the ability to request the resource scheduler to suspend active jobs when power is over subscribed.

Logically, PowSched is comprised of three major functional components: a power *monitor*, a power *scheduler*, and a power *actuator*. The power monitor must run on all managed components and periodically provides consumption measurements to the power scheduler component. The power scheduler uses the consumption measured at runtime and other available information to generate new power allocations. The power actuator must run on all managed components and periodically applies new allocations from the scheduler. Tight coupling between the components is not required. Our job aware PowSched implementation uses UDP communication between the daemons running on the cluster nodes and the power scheduler running on the head node.

1) Node Daemon: The node daemon is responsible for both the monitor and actuator functionality. At regular intervals, node daemons transmit node power consumption to the power scheduler over a UDP socket. A UDP socket to receive new node power caps is also maintained by the daemon. Node daemons use `libmsr` to access power counters and set power caps. To avoid the insecurity of such daemons with root permissions, node daemons can be run with regular user permissions after configuring `msr-safe` [18] on the nodes.

Node daemons utilize Intel RAPL for power cap enforcement. Technologies such as DVFS are more broadly available and can impact power consumption, but software controlled DVFS cannot guarantee power caps are met with high temporal resolution. Intel RAPL is limited in scope to tracking and bounding power for processors and their connected DRAM. Due to the limited scope of Intel's RAPL and lack of other components with similar power control interfaces, the current node daemon implementation only takes processor and DRAM power into account.

2) Central Scheduler: The central scheduler is responsible for interfacing with the job scheduler and dividing the system power cap across the managed nodes. At power scheduling time, the central scheduler determines new node power allocations and messages the node daemons to apply the new allocations. Most of the parameters we would like to explore with our power management platform involve changes to how and when power allocation is done within the central scheduler. The power scheduling decision may involve information received from the node daemons or the job scheduler, and the power scheduler may call back to the job scheduler via the *power* plugin.

For simplicity in communicating with the job scheduler, the central power scheduler is currently launched by the *power* plugin as a thread in the `slurmctld` instance. Keeping the power scheduler inside of the job scheduler's process space greatly simplifies access to the job record and node record data structures. Being in the `slurmctld` process space makes calling SLURM functions simple function calls within the power scheduler, avoiding the upfront cost of identifying the desired SLURM functionality and building RPC mechanisms to wrap them. The tight integration of the power scheduler and job scheduler is not ideal. Future power scheduler iterations are expected to have the power scheduler in a separate process and use SLURM's RPC protocol, sockets, or some other IPC mechanism.

Central power scheduling is done in three phases. The first phase uses recent power measurements and other available data to determine what the updated power allocations should be across the cluster. The second phase saves power by setting the allocation on all nodes having their power reduced. The third phase spends power by setting the allocation of all nodes having their power increased. Attempting to apply all power allocation simultaneously may result in the power bound being exceeded due to messaging latencies [12].

3) Protocol: A simple messaging protocol was introduced to communicate between the node daemons and central scheduler. Several options for reusing existing network protocols were considered and rejected before implementing on UDP. Our protocol is tolerant of some packet loss but assumes the network is high speed and generally reliable. The protocol permits power measurements to be silently dropped, however power allocations must be acknowledged.

SLURM's RPC mechanism was considered early in planning but was rejected in large part due to the node daemons being external processes and the level of effort to extend the RPC interface. REST and other web-service based protocols were rejected due to overheads in processing and network bandwidth. Implementation on TCP was considered, however the reliability and in order guarantees of TCP were not worth the overhead cost for PowSched. Ultimately, we elected to implement our own protocol over UDP.

There is no strong reason to make power measurement messages reliable for power scheduling. If the lost measurement is close to the previously received measurement, which is expected during application phases, then there is little difference between using the lost and previous measurements. If the lost measurement indicates lower power consumption then the new schedule will likely overestimate a node's power need. If the lost measurement indicates higher power consumption then the new scheduler will likely underestimate a node's power need. In all cases the power allocations produced are still safe, though likely less optimal.

Messages used to set power caps, on the other hand, must be acknowledged. Unlike power measurements, unreceived power allocation changes can result in exceeding the system wide power cap. When an allocation is received by the client, the allocation is applied before sending an acknowledgement back to the power scheduler. Setting the cap prior to sending the acknowledgement guarantees that all acknowledged allocations have actually been set. The server sends power allocations in waves with a time out. Each received acknowledgement flips the corresponding nodes `ack` bit in the wave. Unac-

knowledged allocations are resent if the wave time out occurs. Until an acknowledgement is received, a node is assumed to have the highest power allocation sent since the previously acknowledged allocation.

VI. Conclusions

Implementation work is ongoing. As of the time of this writing, preliminary RMAP and PowSched plugins have been implemented. The communication mechanisms between the node selection and power plugins are in place and the current development focus is on policies to explore a spectrum of research questions surrounding job awareness.

SLURM is an open source product with a BSD license. Our extensions leverage the existing plugin architecture and will hopefully be accepted into the main distribution at some point in the future. We are currently working through the release process with our funding organization so that the work can be published to github and made available for general use by the research community.

We have implemented power aware SLURM plugins for experimentally exploring a range of power management strategies for hardware over-provisioned HPC systems. Our work enables the direct comparison of power management strategies using existing hardware platforms. Additionally, our work builds on a robust existing scheduling platform, reducing the distance between state of the art and state of practice solutions.

Acknowledgment

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-701437).

References

[1] *Top 500*, 2016 (accessed August 19, 2016), https://www.top500.org/lists/2016/06/.

[2] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina *et al.*, "The opportunities and challenges of exascale computing–summary report of the advanced scientific computing advisory committee (ascac) subcommittee," *US Department of Energy Office of Science*, 2010.

[3] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra *et al.*, "Top ten exascale research challenges," *DOE ASCAC Subcommittee Report*, 2014.

[4] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski, "Practical Resource Management in Power-Constrained, High Performance Computing," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: ACM, 2015, pp. 121–132. [Online]. Available: http://doi.acm.org/10.1145/2749246.2749262

[5] T. Patki, N. Bates, G. Ghatikar, A. Clausen, S. Klingert, G. Abdulla, and M. Sheikhalishahi, "Supercomputing Centers and Electricity Service Providers: A Geographically Distributed Perspective on Demand Management in Europe and the United States," in *International Conference on High Performance Computing*. Springer, 2016, pp. 243–260.

[6] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple Linux Utility for Resource Management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[7] Y. Georgiou, D. Glesser, and D. Trystram, "Adaptive Resource and Job Management for Limited Power Consumption," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 863–870.

[8] *Slurm Power Management Guide*, 2015 (accessed August 31, 2016), http://slurm.schedmd.com/power_mgmt.html.

[9] B. Rountree, D. K. Lownenthal, B. R. De Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 460–469.

[10] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, "Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 173–182.

[11] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 807–818.

[12] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz, "POW: System-wide Dynamic Reallocation of Limited Power in HPC," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: ACM, 2015, pp. 145–148. [Online]. Available: http://doi.acm.org/10.1145/2749246.2749277

[13] L. Savoie, D. K. Lowenthal, B. R. d. Supinski, T. Islam, K. Mohror, B. Rountree, and M. Schulz, "I/O Aware Power Shifting," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 740–749.

[14] T. Cao, Y. He, and M. Kondo, "Demand-Aware Power Management for Power-Constrained HPC Systems," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 21–31.

[15] D. Ellsworth, T. Patki, S. Perarnau, S. Seo, A. Amer, J. Zounmevo, R. Gupta, K. Yoshii, H. Hoffman, A. Malony, M. Schulz, and P. Beckman, "Systemwide Power Management with Argo," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1118–1121.

[16] N. Gholkar, F. Mueller, and B. Rountree, "Power Tuning HPC Jobs on Power-Constrained Systems," in *International Conference on Parallel Architectures and Compilation (PACT), 2016*. ACM, 2016.

[17] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda *et al.*, "Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 78.

[18] K. Shoga, B. Rountree, M. Schulz, and J. Shafer, "Whitelisting MSRs with msr-safe," in *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*, 2014.