Dynamic Power Management For Hardware Over-provisioned Systems

Daniel Ellsworth*†
*University of Oregon, Eugene, Oregon, USA
†Lawrence Livermore National Laboratory Livermore, California, USA
dellswor@cs.uoregon.edu

***Abstract*—To enable safe operation of hardware over-provisioned computational clusters my work investigates how to engineer distributed power management software in systems where per component power capping hardware is available. My work contributes a formalization for the goal of power management solutions, a generalized model for roughly estimating the performance effect of processor power capping on application runtime, and a fully dynamic power management strategy tested through simulation and experimentation on an existing HPC system.**

I. Problem Statement

My research starts from a practical question: Given technology allowing per component power caps to be set, how can power be safely managed across a shared hardware over-provisioned cluster? Bounded power consumption is the penultimate concern for a hardware over-provisioned system since failure to enforce the global power cap could physically damage the machine. Of solutions that enforce the power cap, good solutions should have minimal complexity, not induce hidden implementation costs, and are broadly applicable. Performance of the system as a whole, in terms of time to solution, is also a critical concern.

II. Problem Background

Through roughly 2005 Dennard scaling allowed the additional transistors predicted by Moore's law to be utilized without significantly increasing the power required to operate the processor. Moore's law predicts that transistor density will grow geometrically. Dennard scaling predicted that power density and, as a side effect, thermal density remains constant. The intersection of Moore's law and the breakdown of Dennard scaling is often discussed in terms of heat dissipation since all power used by the processor is converted to heat.

Two primary techniques to reduce power consumption are 1) power gating and 2) dynamic voltage and frequency scaling (DVFS). Power gating reduces power consumption at the processor level, by turning off parts of the chip not actively engaged in computation, thereby reducing the number of active transistors and associated power losses. DVFS techniques rely on transistor switching power being directly related to switching voltage and frequency, reducing the speed of the processor can decrease the power cost of each active transistor. Modern processor architectures implement both power saving techniques. Use of these techniques lead to time varying power consumption due to the time varying sequence of hardware instructions encountered when running complex applications.

Classically, maximum theoretical power consumption is used when provisioning power for a system, avoiding tripped circuit breakers and errors, even though consumption at the maximum rate is expected to be infrequent. The specific linear instruction sequence encountered by the processor influences which modules will be active within the processor, directly impacting processor power consumption. Since determining the complete instruction sequence without execution is not possible in the general case, computing the time varying power consumption statically is intractable in the general case. Computations backed by preempting operating systems or using interrupts have additional challenges to memory access and instruction stream prediction. Transistor power losses are also affected by heat, and future consumption is therefore dependent on previous instructions executed and hardware cooling infrastructure. Accurate and fine grained a priori estimation of the power consumption within even a single node is likely intractable.

In the context of a computing cluster, the amount of extra power provisioned using maximum theoretical power consumption can be significant. Patki observes that the vast majority of the time less than 70 percent of the provisioned power is actually used by an existing HPC system [4]. The unused provisioned energy is effectively lost. For the same power cost, roughly 30 percent more hardware could have been made available for users. Alternatively, the business could purchase roughly 1 megawatt less power without impacting the majority use of the system. Applying either strategy would result in a *hardware over-provisioned system*, meaning there would be more hardware available than the infrastructure could simultaneously power at maximum consumption.

Existing High Performance Computing (HPC) systems focus primarily on the management of compute resources, typically at the granularity of nodes. HPC users submit work to the computer as jobs and each job contains metadata defining the number of nodes required, the minimum acceptable configuration of each node, and a bound on the maximum time to allow the job to run. The job scheduler holds the job in a queue until sufficient node resources are available to service the job. Job/resource schedulers, such as SLURM, have classically been responsible for resource management.

SC Doctoral Showcase; Salt Lake City, Utah, USA; November 2016

Hardware over-provisioning introduces a new resource management challenge for HPC systems. For resources such as compute, disk, and network, insufficient resource conditions are undesirable but have limited ability to physically impact the system. The power resource is different since a system consuming more power than currently available may result in service interruption for the computer or wider power distribution infrastructure. For hardware over-provisioned clusters, power must not only be managed as a first class resource but also have a strictly enforceable upper bound.

III. Major Research Highlights

A. Power Scheduling Invariant

The power scheduling invariant should be the formalized goal of any power scheduler for a hardware over-provisioned cluster. A safely operating hardware over-provisioned cluster will not exceed the system-wide power cap at any time during operation. The power scheduling invariant relates the system wide power cap to the individual component caps. L, the system-wide power cap, must be greater than the sum of the consumption of all components, c_i, at all points in time.

All components have some power cap, a_i. Since $a_i \geq c_i$, the power scheduling invariant is maintained when the system-wide power cap is greater than the sum of the individual power caps for all components at all points in time.

$$\forall(t,i), c_i^t \leq a_i^t \text{ and } \forall t, \sum_{i=1}^{n} a_i^t \leq L \implies \forall t, \sum_{i=1}^{n} c_i^t \leq L$$

Strategies for power management in hardware over-provisioned systems can be described according to how the power scheduling invariant is maintained. Power management strategies that do not maintain the power scheduling invariant are likely unsafe since they do not provide assurance that the system-wide power cap will not be exceeded. Three main strategies for power management are fully static, job static, and fully dynamic.

Fully static power management strategies set a component power cap when the component is first powered on and the cap remains unchanged for the duration of component operation. Traditionally designed clusters use this strategy, where the power cap is equivalent to the component's maximum theoretical power consumption. The fully static power management strategy is the simplest to implement and prove correct however this strategy has the most chance of wasting power and negatively impacting performance.

Job static power management strategies treat power as a traditional resource, such as disk space or nodes, that can be allocated at job start time. Strategies of this form rely on the scheduler to execute jobs only when sufficient power is available to meet the power estimates for all concurrently running jobs. Power estimates may be provided by the job submitter or may be based on either analytic or machine learning based models. Solutions that rely solely on the estimates and do not set the component caps at job launch time are unsafe since there is no guarantee that the estimate will be obeyed at runtime. Job static solutions are popular in the literature since the approach is based on established high performance computing resource management techniques however the generation of good power estimates is challenging and some power is wasted due to job power consumption varying overtime during execution.

Fully dynamic power management strategies shift power between components during runtime. Strategies of this form rely on runtime introspection to understand the current power consumption across components and an ability to apply changes without pausing running computations. A dynamic power management system may or may not be aware of the mapping from jobs to components, the phase boundaries of computations, or communicate with the application about a change in allocation. Fully dynamic solutions may be able to address power inefficiencies of job static strategies however reasoning about power and performance properties is challenging due to the highly dynamic behavior of the system.

The case of a job static power management strategy where the job is allowed to dynamically shift power allocation between components within the job is interesting. Strategies of this form are important for reasoning through how system-wide power management can operate at extreme scales. For extreme scale, the power management problem can be thought of recursively; the power allocation received from the tier above is the system-wide power cap for the tier below[1].

B. Simulation Model for Power Capped Computations

Real distributed systems have a large amount of variability in performance, measured in terms of wall clock time to solution, making direct study of the impact of power management on performance difficult. Performance variability is known to occur due to switching between the operating system and jobs in addition to contention for shared resources like cache lines and network bandwidth. Small manufacturing differences also cause system variability. Power capping techniques like RAPL can significantly increase variability since the rate of computation per component becomes variable. To study power management strategies with less confounding noise, a simulation model was designed and implemented.

The simulation model is based on coarse experimental observation of power consumption over time and how application performance, measured as wall clock time to completion, changes under progressively lower power caps [2]. Two operating regiemes exist for programs running under a power cap: Unbound, where the cap is higher than the consumption the instruction stream induces, and Bound where an uncapped computation would consume more power than the current cap. Power caps that leave the computation Unbound have no discernible impact on time to solution. Power caps that leave the computation Bound cause the runtime to be dilated by and amount related to the difference between the uncapped consumption and the power cap. For applications with significantly different power consumption across phases, the effect of dilation can be seen only in the phase that would consume above the power cap.

Experiment	Runtime	Stddev	Improvement	kj Alloc	Stddev	kj Used	Stddev
115W static	278.26	9.57		8191.85	281.75	4007.80	97.99
115W dynamic	276.24	4.84	0.7%	5474.75	52.56	3977.02	36.74
90W static	284.63	3.20		6571.76	72.75	3984.68	30.32
90W dynamic	277.13	5.04	2.6%	5339.11	66.21	3979.78	47.356
70W static	323.83	4.90		5829.02	86.82	3904.29	34.08
70W dynamic	278.02	4.97	14.1%	4638.32	68.91	3984.80	37.77
50W static	401.76	5.47		5178.29	72.59	3937.65	37.52
50W dynamic	371.92	13.23	8.7%	4562.48	124.44	4015.64	79.36

Fig. 1. 128 nodes, 16 nodes workloads per workload, 10 runs, same workload for all runs.

Programs are modeled as functions of power consumption over time; at time t a program would consume $w(t)$ watts. Over an interval $w(t)$ represents the energy loss incurred by active transistors on an uncapped processor over that interval. Since the energy is lost by active transistors, which do the work of the instruction stream, abstractly a relationship exists between $w(t)$ and the instruction work a program induces on a processor.

$$I_p = \int_s^e \sqrt{\frac{w_{(t)} - S_-}{S_+ - S_-}} \mathrm{d}t \quad , \quad E_p = \int_s^e w_{(t)} \mathrm{d}t$$

Processors are modeled as functions yielding instruction work possible at a given power cap. Some minimum power is consumed by the processor due to static losses and background system activity, at this minimum power the processor can conduct no work for a running application. At the manufacturer's specified maximum power dissipation, the processor can do the maximal amount of instruction work on behalf of an application.

$$I_s = \int_s^e \sqrt{\frac{b - S_-}{S_+ - S_-}} \mathrm{d}t \quad , \quad E_s = \int_s^e b \mathrm{d}t$$

While there are programs active in the simulation:

- If the power cap over the interval $s \rightarrow e$ exceeds the power consumption induced by the program:
  - Advance the program's time by $s \rightarrow e$
  - Consume the energy based on the program's consumption
- Otherwise:
  - Compute the instruction work the processor can service over interval $s \rightarrow e$
  - Compute how long it would take for the program to induce that much instruction work on an uncapped processor
  - Advance the program's counter by the computed duration
  - Consume the energy based on the cap and $s \rightarrow e$

The simulation is useful for understanding power management system behaviors since the characteristic application performance under progressively lowered power caps is produced by the simulator. Simulator output is not intended to be high fidelity and the sacrifice is intentional. Results from the simulator are deterministic up to numeric error during the integral computations and much less noisy than experimentally generated data. For exploring dynamic power management, simulated introspection and power control signal latencies can be controlled. The model and simulation are also predictive since program and processor models can be used for which there is no existing experimental data.

C. PowSched Solution

PowSched is an effective and simple system for fully dynamic power management that operates with no awareness of jobs within the system. Over each scheduling interval PowSched centrally collects power consumption from each socket in the system, computes new per socket allocations based on the amount of unused power per socket, and applies the newly computed allocations. The solution uses no application or hardware specific models and does not attempt to learn over time. Experimentally, on existing hardware, PowSched has achieved a 14% performance improvement over a fully static power schedule for the same workload and system-wide power cap (figure 1).

Power shifting is done in PowSched based on the difference between power consumption and allocation over the scheduling interval [3]. PowSched sets per component allocations such that the power scheduling invariant is guaranteed to be maintained ($L \geq \sum a_i$). A component's allocation is reduced when the component power consumption falls beneath a threshold from the existing allocation ($a_i - \epsilon_- < c_i$). Unallocated power then becomes available for components appearing to need additional power. A component's allocation is increased when the component power consumption is within a threshold of the present allocation ($c_i + \epsilon_+ > a_i$).

There is an implicit assumption by PowSched that the concurrently scheduled work will fit within the system-wide power cap. When the assumption is met, no job completions should be delayed due to power limitations. Given the consistent level of aggregate power consumption observed in existing uncapped clusters the assumption is reasonable [4]. Fairness becomes a concern when the power scheduler must degrade job performance to maintain the system-wide power cap. In the event that all power is allocated and some components appear in need of additional power, PowSched will take power from all components and attempt to converge toward a equal power allocation for all components.

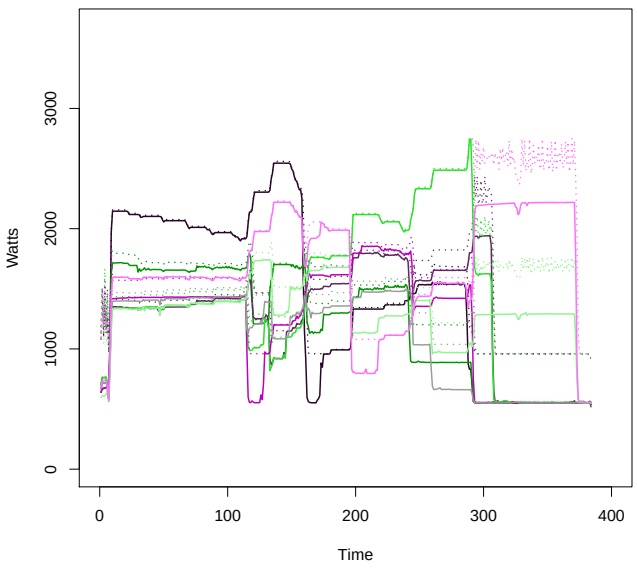

Fig. 2. Power consumption and allocations showing opportunistic shifting jobs consuming little power to jobs consuming much power. Convergence behavior can also be seen when PowSched detects the system is power bound.

Experimental results have shown good performance with a 70 watts per socket cap on hardware with a theoretical maximum consumption of 115 watts per socket. By graphing the power consumption per node correlated with the mapping of nodes to jobs, the power shifting and convergence behavior of PowSched are visible as well as the identification of when and on which nodes high and low power consuming jobs are running (figure 2). Experimentally observed runtimes across power caps are within jitter when not faster than fully static configurations (figure 1).

Running PowSched in simulation produces clear results indicating when PowSched can be expected to provide performance improvements [2]. For good performance, the system-wide power cap must be low enough that an equal power cap across components is too low for some job phases. Additionally, the consumption phase changes of the work in the system should not be constructively time aligned. PowSched is unable to provide any improvement in performance if all jobs increase and decrease consumption together. Ideally the sum of the per component consumption of concurrently scheduled work is just under the system-wide power cap and consumption phase changes occur destructively, any component increasing consumption is matched by a component decreasing consumption at every time step. In the space between ideal and worst case alignments, PowSched is able to return some performance improvement.

Even though PowSched uses centralized control, the solution appears to scale well to large component counts. Consumption is monitored and reported to a single node, which unilaterally makes the power scheduling decision, and broadcasts the new allocations to the controlled nodes. Overhead for the operations is low since the central scheduler need only receive and send a single floating point value per configurable component. Scheduling decision time is linear in the number of nodes and done using a small number of scans. 256 node experiments have been conducted showing PowSched induces negligible overhead. An experiment investigating communication and computation scaling for up to 512k nodes indicated that PowSched could complete a scheduling interval in under 500ms using existing interconnects [2].

D. Job Awareness

To date, the work on PowSched has assumed the power scheduler has no knowledge of jobs or job characteristics. Intuitively better power scheduling decisions seem possible if some information is known. One obvious improvement from additional job information would be the ability of the power scheduler to allocate power in alignment with the business value of different jobs. Other improvements might be possible through interactions with the job scheduler to apply hints regarding estimated power consumption to guide upper and lower job power bounds. Work over the coming year will be exploring the trade-off space between job agnostic and job aware dynamic power scheduling.

Exploration of the trade-off space will be done in simulation and experimentally. The simulation platform job queue is being augmented with additional features to support mapping simulated sockets to their associated job id. For experiments on existing hardware, a SLURM power plugin is being written to facilitate communication between the job and power scheduling logic. A SLURM select plugin is also in development to facilitate direct comparisons between job static and fully dynamic techniques.

Acknowledgment

This work would not have been possible without the mentorship of Allen Malony, Martin Schulz, Barry Rountree, and Tapasya Patki. 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. Work at the University of Oregon was supported by the DOE Office of Science, through a Sub-Contract No. 3F-32643 from the University of Chicago, Argonne, LLC (as operator of Argonne National Laboratory), under Prime Contract No. DE-AC02-06CH11357. LLNL-CONF-705518

References

[1] D. Ellsworth, T. Patki, S. Perarnau, et al. Systemwide power management with argo. In *Workshop on High-Performance, Power-Aware Computing*, 2016.

[2] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Dynamic power sharing for higher job throughput. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015.

[3] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Pow: System-wide dynamic reallocation of limited power in hpc. In *International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, New York, NY, USA, 2015. ACM.

[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 *High-Performance Parallel and Distributed Computing*. ACM, 2015.