LLNL-CONF-669276

POW: System-wide Dynamic Reallocation of Limited Power in HPC

D. Ellsworth, A. Malony, B. Rountree, M. Schulz

April 1, 2015

HPDC 2015
Portland, OR, United States
June 15, 2015 through June 19, 2015

POW: System-wide Dynamic Reallocation of Limited Power in HPC

Daniel A. Ellsworth, Allen D. Malony
University of Oregon
Eugene, Oregon, USA
{dellswor,malony}@cs.uoregon.edu

Barry Rountree, Martin Schulz
Lawrence Livermore National Laboratory
Livermore, California, USA
{rountree4,schulzm}@llnl.gov

ABSTRACT

Current trends for high-performance computing systems are leading us towards hardware over-provisioning where it is no longer possible to run each component at peak power without exceeding a system or facility wide power bound. In such scenarios, the power consumed by individual components must be artificially limited to guarantee system operation under a given power bound. In this paper, we present the design of a power scheduler capable of enforcing such a bound using dynamic system-wide power reallocation in an application-agnostic manner. Our scheduler achieves better job runtimes than a naïve power scheduling approach without requiring a priori knowledge of application power behavior.

Categories and Subject Descriptors

D.4.m [**Operating Systems**]: Miscellaneous

Keywords

RAPL; hardware over-provisioning; HPC; power bound

1. INTRODUCTION

Scalable parallel applications have been the driving force behind the evolution of large-scale parallel systems with ever-increasing demands for processor, memory, and network performance. The evolution over the past decade has followed a “horizontal” scaling strategy to increase floating-point operations per second (*flops*) and I/O operations per second (*iops*) by adding more of the latest hardware. However, powering a massive cluster at the maximum simultaneous power draw of all hardware components is a major challenge, yet often unnecessary since few applications are able to fully exploit all components at peak capacity.

An alternative power strategy is *hardware over-provisioning*, where more hardware is available than can be powered at maximal draw at any time [4]. In this case, power provisioning and system scale can be designed for the common case, but mechanisms are required to prevent the system from exceeding the predetermined maximal power draw. New technologies, such as Intel’s Running Average Power Limit (RAPL), are a key enabling technology for hardware over-provisioning. However, while RAPL provides the necessary software configurable and hardware enforced power cap per CPU socket, an additional power distribution algorithm to spread the available power across the system is still required.

In this work, we present a dynamic power scheduler that monitors power consumption and reallocates power across a cluster. The power control system enforces the global power bound without requiring integration with the job scheduler. Using a simple heuristic our power scheduler reclaimes wasted power in the overall system to components restricted by their current power bound. This leads us to a dynamic power control system that enforces a global power budget while being completely opaque with respect to the particular applications in the workload, their power and performance characteristics, and their mix.

HPDC’15, June 15–20, 2015, Portland, Oregon, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3550-8/15/06 ...$15.00.
http://dx.doi.org/10.1145/2749246.2749277

2. APPROACH

Our work targets large-scale high-performance computing (HPC) systems, primarily with an eye to future exascale platforms. HPC systems represent a substantial capital investment and are typically shared batch-scheduled resources. An HPC system is composed of many compute *nodes*, each with a number of processing elements, including CPUs and accelerators. Users of the system typically submit *jobs* with a desired number of nodes to a job scheduler where each job is queued. The scheduler will schedule a job to run when an adequate number of nodes become available. We will call a subset of the nodes assigned to a job a *partition* or *enclave*, and will assume that any particular node is a member of only one enclave at a time.

The HPC environment is highly parallel and concurrent. User jobs are typically multi-node highly-parallel applications and several jobs will run simultaneously on an HPC system. A job’s start time is determined by node availability and a job’s end time is based on the actual runtime (or maximum time allocation) of the job. Although the HPC machine is *space-partitioned*, in that each job has its own processing resources, certain shared resources (e.g., network, file system, power) are used by concurrently executing jobs, potentially impacting the runtime behavior across jobs.

One of the major challenges in the move from current petascale to future exascale computation is increasing computational power within realistic electrical power consumption. The current approach of designing power systems to

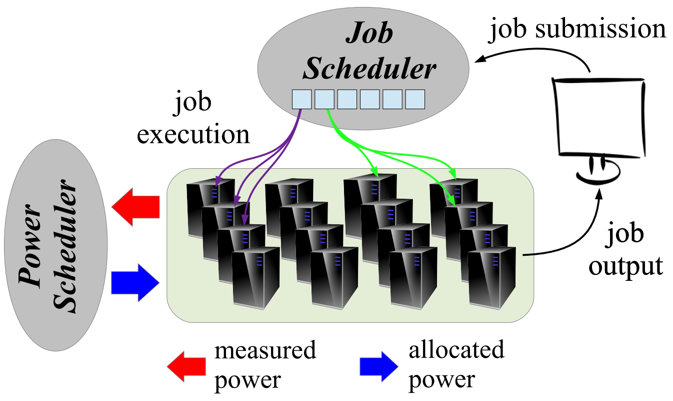

Figure 1: High-level model of system interactions

L	System-wide power limit
n	Number of sockets
t	A timestamp
c_i^t	Power consumed by socket i at time t
a_i^t	Power allocated to socket i at time t
w_i^t	Unused portion of socket i's allocation at time t
C_{min}	Min observable socket consumption
C_{max}	Max observable socket consumption
A_{min}	Min allocation for a socket according to the spec
A_{max}	Max allocation for a socket according to the spec

Table 1: Symbols use in the model

sustain peak power at all times, even though few jobs consume energy at that rate, is therefore unrealistic. Hardware over-provisioning is likely the only way to achieve the increase in computing power while maintaining the power budget, but requires new approaches to distribute the available power to components and to enforce that components stay within their assigned power limits.

We assume future hardware platforms will support an interface with properties similar to Intel's *Running Average Power Limit (RAPL)*. In current systems, components supporting RAPL can enforce a configurable maximum rate of energy consumption over a sliding temporal window. The particular techniques used to enforce the limit are selected and implemented completely by the hardware. The RAPL interface in our testbed uses *model-specific registers (MSRs)* (accessible via *libmsr* [3]) to allow software to interact with the hardware power management facilities.

A mechanism like RAPL alone, however, is insufficient for running in an over-provisioned environment, since it only enables the setting of power bounds for individual components. A global power scheduler is needed to control the individual power bounds for all components and ensure that the total sum of all bounds is below the total system bound. Exceeding the total system bound could damage the HPC cluster or the supporting power infrastructure.

Figure 1 shows a high-level view of the interaction between a potential power scheduler and an HPC cluster. The job scheduler is responsible for assigning jobs to hardware resources as well as starting and stopping the jobs. The power scheduler is solely responsible for analyzing power measurements from the cluster and providing updated power allocations to all cluster components. The HPC cluster itself is primarily concerned with executing jobs from the scheduler, but also provides the integrated infrastructure for power measurement and control used by the power scheduler.

2.1 Power Model

The system-wide power scheduler has the primary objective of enforcing a global power limit, L. We can think of the HPC system as having an infinite amount of energy but having a global maximum limit to the instantaneous rate at which energy can be used[1]. Power-optimization and energy-aware techniques reduce the energy consumed [1, 7, 6], often by reducing the power while maintaining the runtime, allowing more of the hardware over-provisioned system to be used concurrently. These techniques do not provide a guarantee that global rate of energy consumption remains within a fixed bound. Reduced energy consumption and optimal runtimes are secondary objectives for a power scheduler charged with enforcing the global power limit in a hardware over-provisioned system.

A global power limit L is set by facility limitations or administrative policy to protect the power infrastructure from damage due to exceeding capacity. A system is modeled as a set of n sockets. Every socket i has a power consumption, c_i, and a power allocation, a_i. The delta between c_i and a_i is the *wasted*[2] allocation and will be noted as w_i. It is assumed that the hardware enforces $c_i \leq a_i$ or equivalently $a_i = c_i + w_i$. Thus, the total power allocated to the system is $\sum a_i$ and the total power consumption is $\sum c_i$. Further, due to the hardware enforcement, $\sum c_i \leq \sum a_i$.

In the following sections, we will use the intuition that application runtime is roughly the same for any a_i such that $a_i > c_i$. Runtime should only be impacted when a_i is less than the amount an application would consume if there was no power bound. This conclusion is consistent with Fukazawa et al. [2] and our own experiments, which have been omitted for length.

2.2 Static Scheduling

A *static* power scheduler makes a decision about how to schedule power prior to the job launch. A naïve scheduling strategy would be to allocate an equal amount of power to each socket, $a_i = \frac{L}{n}$, over the lifetime of the machine. Since $\sum c_i \leq \sum a_i$, trivially this strategy maintains $L \geq \sum c_i$. Two existing systems at the Lawrence Livermore National Lab (LLNL) use this strategy presently.

While meeting the technical requirement of enforcing a global power bound, the naïve static strategy is expected to under perform. There are two reasons for this. First, prior work has shown a non-linear relationship between power allocation and application performance [5]. Setting each socket to the same level could degrade performance if that level is too low. Second, power consumption could be different on each socket used. A static, equal power setting for all sockets could disrupt performance non-uniformly.

A more refined static power scheduler could attempt some optimization of power distribution if it was aware at scheduling time of an application's characteristic power consump-

[1]Energy and power are separate but related ideas. Energy is typically measured in joules. Power is a rate typically measured in watts, representing joules per second.

[2]The power is wasted in that power was allocated to the system but not used by the system.

tion. For instance, if w_i reflects performance behavior under an allocation a_i, then w_i could be used as a basic metric for optimization. Rather than allocating an equal amount of power to each socket, the static scheduler could allocate an equal amount of wasted allocation, w_i, to each socket. The allocation per socket for such a scheduler can be computed using $a_i = c_i + w_{avg}$ where $w_{avg} = \frac{1}{n}(L - \sum c_i)$.

For the more refined static approach, the scheduler must know *a priori* the corresponding c_i and w_{avg} values in the system. The behavior of a job can change based on the parameters used for execution and there is also an expectation of greater uncertainty in behavior as systems are scaled due to increasing runtime and interactions with other jobs. For long lived clusters, where numerous jobs of various sizes asynchronously enter and exit the system, $\sum c_i$ across the system is expected to vary greatly over time as jobs enter and leave the system. Even within a single job, different phases may consume energy at different rates. Knowledge of per socket power consumption in advance of execution is therefore not feasible in the general case.

2.3 Dynamic Scheduling

Static power scheduling at job launch time cannot maintain w_{avg} across the full machine in the presence of dynamic job power consumption and missing knowledge of future jobs. A dynamic approach to power scheduling is likely required to respond to the dynamic power consumption observed at runtime. Rather than attempting to set a_i once at job start time, a dynamic scheduler can periodically adjust any a_i in the system, even when there is an active job running on the socket.

Extending the model to include time, the scheduler must guarantee for all times t that $L \geq \sum c_i^t$. A basic dynamic scheduler strategy may assume that the power consumption of a running job remains fairly consistent over time, represented by the heuristic $c_i^t \approx c_i^{t-1}$. At time t, the scheduler can know the values c_i^{t-1} and a_i^{t-1}, as reported by the socket, as well as L. The updated per socket allocation can be computed as $a_i^t = c_i^{t-1} + w_{avg}^{t-1}$.

Using the formulation above, a dynamic power scheduler can maintain w_{avg} without any control of the job scheduling. If the scheduler is able to maintain $w_{avg} > 0$ then all applications are expected to complete with their unbounded runtime by the intuition that runtime is not degraded when $a_i^t > c_i^t$. The power scheduler only requires c_i^t and a_i^t for all sockets as input to set all a_i^{t+1} during runtime.

Up to this point in the discussion, there has been an assumption that there is sufficient power to run all scheduled jobs at the optimal power consumption, $c_i^t < a_i^t$ for all i and t. This assumption requires a job scheduler that is guaranteed to never oversubscribe power. Due to the challenges discussed for static power scheduling, requiring the job scheduler to produce a schedule that never oversubscribes power and can consume the full system wide power allocation is not practical.

A dynamic scheduler reading where $c_i^t = a_i^t$ could indicate that the power is set to exactly what the application using the socket can consume. Alternatively, $c_i^t = a_i^t$ could indicate that a_i^t was too low and that the hardware reduced consumption on the socket, degrading application performance. The responsiveness of a dynamic power scheduler to increased consumption, using the formulation in this section, is expected to be impacted by both the scheduling interval and the per socket wasted power allocation due to the assumption $c_i^t \approx c_i^{t+1}$ and hardware enforcement of $c_i^t \leq a_i^t$.

3. DESIGN

POWsched is a *dynamic* power scheduler based on the model and approach discussed in the previous section. Scheduling decisions in POWsched are per socket and are completely agnostic with respect to job, enclave, and node the socket is associated with. POWsched maintains a system-wide power bound without job scheduler coordination using only per socket observed power consumption to guide power scheduling across a cluster.

Pseudocode for the scheduler is provided in Algorithm 1. The scheduling task is performed in three phases during each scheduling interval. In Phase 1 POWsched collects recent consumption readings from all sockets. In Phase 2 power is greedily recovered from the existing allocations for later distribution. In Phase 3 additional power is given to sockets that may be able to use the power. At the end of Phase 3, POWsched sleeps the remainder of the scheduling interval.

Separation of power allocation into two phases is needed to guarantee that the system wide power limit is never exceeded due to communication delays. Recall that $a_i^t \leq L$ must be maintained for RAPL to successfully enforce $c_i^t \leq L$. Assume $a_0^t + a_1^t = L$. If the scheduler computes $a_0^t > a_0^{t+1}$ and $a_1^t < a_1^{t+1}$ and sends a_0^{t+1} and a_1^{t+1} at the same time, communication delays might cause socket 1 to update the allocation before socket 0. For a short interval the allocated power will be $a_0^t + a_1^{t+1} > L$, which is a violation of the system power bound. POWsched must be certain that all sockets receiving a lower allocation have been updated before any sockets receiving a higher allocation are updated.

POWsched does not compute w_{avg}. A target w_i is used to account for the measurement jitter and greedily reclaim power from under consuming sockets. POWsched assumes the system is oversubscribed and steals a percentage of the allocation for each socket allocated more than the system wide average per socket allocation ($a_i > \frac{L}{n}$) when no power can be reclaimed and very little surplus power is available. When adjusting allocations up, POWsched divides the surplus power evenly across the sockets consuming near their current allocation. When power is abundant, the allocation up behavior is expected to result in a lot of wasted power that can then be greedily collected in the next scheduling round. When power is scarce, the allocation up and power stealing behavior will eventually converge at a fair allocation across all sockets.

We implemented POWsched in C using libmsr to access the RAPL MSRs and MPI for collective communication. In our experiments with POWsched on the cab cluster at LLNL, POWsched is deployed as a separate process coresident with the actual application workload[3]. This strategy allows us to use existing system setups within the constraints of the existing job scheduler. In future systems power scheduling is likely to be provided as part of the system stack as part of a global operating system. Our preliminary results indicate overall runtime can be improved over naïve power scheduling.

4. CONCLUSION

[3]A workload in our experiments consist of several concurrent jobs.

Algorithm 1 POWsched logic in pseudocode

```
$q \leftarrow$ target $w_i$
$C$ stores $\{c_0, \cdots, c_{n-1}\}$
$A$ stores $\{a_0, \cdots, a_{n-1}\}$
$M$ stores $\{m_0, \cdots, m_{n-1}\}$
numdown $\leftarrow$ count of nodes yielding power
interval $\leftarrow$ scheduling interval
reclaimfactor $\leftarrow$ power to reserve when stealing

procedure MAIN
    while True do
        GETREADINGS                                  ▷ Phase 1
        ALLOCDOWN                                    ▷ Phase 2
        ALLOCUP                                      ▷ Phase 3
        sleep rest of interval
    end while
end procedure

procedure GETREADINGS
    for all sockets do
        Update $c_i$ with the current reading
    end for
end procedure

procedure ALLOCDOWN
    numdown $\leftarrow 0$
    for all sockets do
        if $c_i < a_i - q$ then
            Update $a_i$ to $max\{c_i + q, A_{min}\}$
            numdown $\leftarrow$ numdown + 1
            Update $m_i$ to False
        else
            Update $m_i$ to True
        end if
    end for
    if numdown$= 0$ and $\sum a_i + n \geq L$ then
        for all sockets do
            if $a_i > \frac{L}{n}$ then
                $a_i \leftarrow a_i - (a_i - \frac{L}{n}) \times (1-$reclaimfactor$)$
                $m_i \leftarrow$ True
            end if
        end for
    end if
    for all sockets do
        Set the socket to limit $a_i$
    end for
end procedure

procedure ALLOCUP
    $u \leftarrow \frac{(L - \sum a_i)}{n - \text{numdown}}$
    for all sockets do
        if $m_i$ then
            $a_i \leftarrow min\{a_i + u, A_{max}\}$
        end if
    end for
    for all sockets do
        Set the socket to limit $a_i$
    end for
end procedure
```

We have described a system-wide dynamic power scheduler that enforces a global power limit on an HPC system without requiring application specific profiling or application modification. POWsched monitors power consumption during the execution of multiple simultaneous applications and reallocates power to individual node sockets based on simple heuristic. We expect POWsched to out perform static fixed power allocation without a priori application analysis due to the ability to dynamically reallocate power from under consuming sockets to power bound sockets. Future work will focus on performance evaluation and assessing scalability issues in anticipation of next-generation exascale systems.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy (DOE) Lawrence Livermore National Laboratory. Office of Science, under Award number DE-AC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research (LLNL-CONF-656877). Work by the University of Oregon is 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.

6. REFERENCES

[1] M. Bambagini, M. Bertogna, M. Marinoni, and G. Buttazzo. An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 3–12. IEEE, 2013.

[2] K. Fukazawa, M. Ueda, M. Aoyagi, T. Tsuhata, K. Yoshida, A. Uehara, M. Kuze, Y. Inadomi, and K. Inoue. Power consumption evaluation of an mhd simulation with cpu power capping. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 612–617. IEEE, 2014.

[3] L. L. N. S. LLC. libmsr. `https://github.com/scalability-llnl/libmsr`.

[4] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *27th ACM International Conference on Supercomputing*, pages 173–182. ACM, 2013.

[5] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops (IPDPSW),*, pages 947–953. IEEE, 2012.

[6] 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 *23rd ACM International Conference on Supercomputing*, pages 460–469. ACM, 2009.

[7] A. Tiwari, M. Laurenzano, J. Peraza, L. Carrington, and A. Snavely. Green queue: Customized large-scale clock frequency scaling. In *Second International Conference on Cloud and Green Computing (CGC)*, pages 260–267. IEEE, 2012.