Hypnos: Unobtrusive Power Proportionality for HPC frameworks

Arka Bhattacharya
David E. Culler

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-29
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-29.html

April 15, 2014

Copyright © 2014, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Hypnos: Unobtrusive Power Proportionality for HPC frameworks

Arka Bhattacharya[1] and David Culler[1]

[1]University of California, Berkeley

Abstract

The proliferation of large High-Performance Computing clusters executing computation-intensive jobs on large data sets has made cluster power proportionality very important [13]. Despite publicly available traces showing that many clusters have a low average utilization, existing power-proportionality techniques have seen low adoption, a major reason being that these techniques require modifications to the existing cluster software and network stack, and do not address the reliability concerns that may arise during the course of server power-cycling.

We present *Hypnos*, a defensive power proportionality system which is unobtrusive, extensible and gracefully handles possible server software and hardware failures which may occur during server power-cycling. We deployed Hypnos on a 57-server production cluster. From a 21-day run, we obtained a 36% energy saving in spite of multiple server and network failures.

1 Introduction

The use of High Performance Computing (HPC) in diverse fields ranging from machine learning and statistical physics to weather simulation, has led to a massive increase in energy consumption of such clusters [13]. Clusters are generally provisioned for slightly more than peak estimated load. The slight over-provisioning helps the cluster handle unforeseen load spikes. However, recent research has shown that in some clusters, the peak load occurs rarely [13], resulting in the average load on the servers being far below the peak. The implication is that a lot of servers do no work, but consume energy in their idle state. One of the key tenets of system design is: in making a design trade-off, favor the frequent case over the infrequent case. The systems community, for long, has focused on optimizing for performance — making the common case fast [25]. But the common case for many of the servers in such clusters is *doing nothing*; that is, being idle. This common case of doing nothing consumes power because servers are not power proportional (*i.e.,* power consumption is not directly proportional to utilization [15]). If we are to design for the common case from a power perspective, we need to "do nothing well".

Building power proportional clusters comprising of servers which are not power-proportional is a well-studied problem ([14, 17, 18, 19, 22, 20, 21]), and has been shown to provide large energy savings. In spite of the large body of existing work, cluster power proportionality has seen low adoption, especially in clusters in academic settings, where the main job of the small IT support staff is to ensure cluster uptime[1]. The large emphasis on maintaining cluster uptime disincentivizes cluster administrators from deploying new power management software which maybe need changes to software configurations, change user job submission procedures, or fail in unpredictable ways. Most prior research in power-proportionality often require modifications to the already complex cluster frameworks and configuration scripts (§2.2). Also, some prior work does not deal with the possibility of software and hardware failures which can arise when servers are regularly power cycled [24], making these solutions non-viable for deployment on such clusters.

In this paper, we tackle the challenge of implementing power proportionality unobtrusively in a HPC cluster with a shared centralized filesystem. An unobtrusive power proportionality system would entail no changes be made to the existing cluster software or network stack. Thus, the new power management system would have to use only the standard interfaces exposed by the cluster's existing job management framework to obtain server state information and infer the cluster's scheduling logic .

Even though most commercial HPC cluster frameworks are tolerant to server failures during normal operation, frequent power-cycling of servers based on inferred knowledge leads to failure scenarios that may

[1]Solutions such as [21] have been deployed in production clusters, but mainly at big companies with large infrastructure support

confuse an unobtrusive power proportionality system. For instance, a job may be stuck in a queue due to per-user limits implemented inside the cluster's job management framework. In such a case, the power proportionality algorithm should not power up a new server to serve the queued job. Figuring out the difference between this scenario, and one where a job is queued up due to a shortage of powered-up servers is a challenge for an unobtrusive system. Also consider a scenario in which the power proportionality algorithm wants to power down an idle server. Suppose the server does not shut down properly due to the "power-down" command getting dropped over the network, or some software process on the server keeping the server from shutting down, leaving the server in an undesirable state. A "wakeup" command sent to this server at a later time would not have any effect as neither had this server shut down, nor might it be in a desirable state to run new jobs. The unobtrusive system should be able to detect this scenario and act accordingly.

We present *Hypnos*, a defensive meta-system for obtaining power-proportionality in production clusters running an HPC framework with a shared centralized filesystem. Hypnos has three main design principles - unobtrusiveness, fault tolerance and extensibility. Hypnos attains unobtrusiveness by sitting on top of the master server and using existing HPC framework interfaces to infer cluster state and server failures. Hypnos maintains a state-machine for each server, and checks for failures when a server transitions from one state to another. If a failure is inferred, Hypnos places the faulty servers in a separate flagged state, and exposes this information to the power management algorithm, which then re-assesses the set of servers to be woken up or powered down. The modular design of Hypnos enables its easy adaptation to different HPC frameworks by simply changing the framework specific parser. We evaluate Hypnos by deploying it on a production cluster running the HPC framework - Torque [10] in an academic environment. Hypnos was able to achieve a 36% reduction in energy consumption (compared to an optimal of 37.5%) while circumventing over 1500 network and software faults over a 21-day deployment.

2 Background

2.1 The Relevance for Cluster Power Proportionality

Servers are not power-proportional and can consume 20-80% (Figure 1) of their peak power even when idle, implying that even largely idle clusters can consume copious amounts of energy. Even though the processor is becoming more power proportional through mechanisms such as frequency scaling and clock gating, other components (such as memory, IO, etc) and peripherals (fans, etc) are still non-power proportional [22]. Sleep states, which are common in mobile devices, are still not widely available in servers [16]. Trends do show that the server idle power consumption is decreasing. However, it will a long time before legacy servers are completely replaced by completely power proportional servers, making cluster-level power proportionality the only option to reduce the energy wastage.

Consider a cluster with n servers, each consuming power P_{peak} at peak utilization and P_{idle} when it is idle. To get a very simple intuition into the energy savings possible due to power-proportionality, let us assume that each non-idle server is operating at its peak utilization, and the cluster is not over-provisioned [2]. Let the average number of servers running jobs over a time period T be n_{avg}.

The energy consumed by an ideal power-proportional cluster (E_{pp}) over the time period T would be

$$E_{pp} = n_{avg} \times P_{peak} \times T$$

The energy consumed by a non-power-proportional cluster [3] (E_{npp}) would be

$$E_{npp} = E_{pp} + (n - n_{avg}) \times P_{idle} \times T$$

Hence, the percentage energy savings obtained by converting a completely non-power proportional cluster to an ideally power-proportional one would results in en-

[2] We consider the peak load in the trace to be the provisioned capacity

[3] a cluster which keeps all servers powered on the entire time

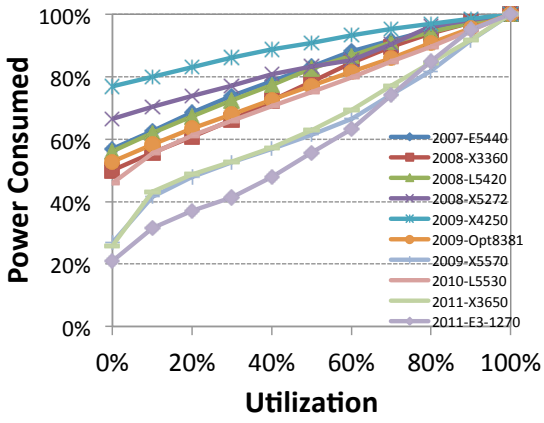


Figure 1: Performance vs Power characteristics of different servers according to spec power_ssj2008 benchmark

ergy savings ($\%E_{savings}$)given by

$$\%E_{savings} = \frac{(n - n_{avg}) \times P_{idle} \times T}{E_{npp}} \times 100$$

If we denote the ratio $\frac{Pidle}{P_{peak}}$ by s[4], and the ratio of the total number of servers in the cluster to the average number of servers utilized over the time period as $f_{peak/avg}$, we can simplify the expression $\%E_{savings}$ as

$$\%E_{savings} = \frac{1}{\frac{1}{(f_{peak/avg}-1)\times s} + 1} \times 100$$

Three interesting points spring out of this simple analysis. First, if individual servers were power proportional (*i.e.,* $s = 0$), there would be no possible energy savings because the cluster would already be completely power-proportional. Second, the higher the idle power of a server in comparison to its peak (s), the higher is the energy savings possible. Finally, the most important factor controlling the amount of energy savings is the peak to average ratio ($f_{peak/avg}$) of cluster utilization. The higher the value of $f_{peak/avg}$, the more is the incentive to deploy cluster-wide power proportionality mechanisms. A high value of $f_{peak/avg}$ indicates that the peak load the cluster is provisioned for occurs rarely, and a much lower average load keeps most of the servers spinning idly.

Table 1 shows the $f_{peak/avg}$ and $\%E_{savings}$ of some publicly available HPC traces from clusters in an academic setting, assuming $s = 0.5$ (i.e the server consumes 50% of its peak power when idle). From the analysis, it is evident that cluster power proportionality need not benefit all clusters. For instance, the NERSC clusters' average utilization closely matches its provisioned capacity and has very few idle servers to power down. The remaining clusters display considerable amount of idleness and energy wastage. There is, thus, a strong case for implementing power proportionality in such under-utilized HPC clusters with $\%E_{savings}$ ranging from 38%-82%.

2.2 Prior Work and Feasibility of Existing Solutions

Despite the potential for significant energy savings and the large body of existing research in this area, cluster power proportionality has not been widely deployed among HPC clusters. Through private conversations with system administrators, the primary concerns seem to be the obtrusiveness and reliability of the power management system.

[4] s quantifies the server non-power proportionality (or *local* power proportionality)

Table 1: Statistics from different HPC clusters, whose utilization is archived in [9]. The degree of under-utilization of a cluster can be determined by its $f_{peak/avg}$ value. The notation used in the table is same as that in Section 2.1

Cluster	Avg Util.	$f_{peak/avg}$	$\%E_{savings}$
LCG	35%	2.9	49%
Grid 5000	17%	6	71%
Nordu Grid	10%	10	82%
AuverGrid	35%	2.9	49%
Berkeley PSI	45%	2.2	37.5%
NERSC Franklin	93%	1.08	3.8%
NERSC Hopper	88%	1.14	6.5%

Existing HPC Power Managers: There exists quite a few existing power managers available for HPC job-sharing frameworks. The licensed Green Computing scheduler from Moab [1] integrates power management into its framework and requires configuration files to be modified to enable power proportionality. System administrators are, thus, compelled to update the cluster configuration files every time the power management features need to be enabled or disabled. Also, reconfiguring a cluster setup increases the risk of misconfigurations and failures. Rocks-Solid [8] has a feature to enable power saving for a clustering running ROCKS [7], but deploys a very naive algorithm which does not power down any server if even as long as there exists queued jobs. Also, it wakes all servers up until a queued job is executed. This is clearly inefficient because a job can be queued for fairness or other purposes, and waking up servers may not enable it run. It is, thus, unreliable. R-energy [6] is a remote energy management tool which implements power proportionality only for IBM servers which support IBM EnergyScale technology.

Prior research in Power Proportionality: One way to reduce operating expenditure in a data center is to reduce the amount of non-sever related power expenditure in a data center. Efforts in reducing the Power Utilization Efciency (PUE) of large scale data centers[5], has resulted in the typical values reducing from 2 or greater, to state-of-the-art facilities having PUEs as low as 1.12 [2], *i.e.,* much closer to the ideal value of 1.0.

For data centers with low PUE values, the next challenge is to reduce the amount of computing work done per unit energy. Existing research has looked into reducing the energy consumed by individual servers (*i.e.,* reducing idle energy consumed by servers) by using low-power processors [12] or by introducing sleep states

[5] PUE is the ratio of total data center consumption to that consumed by the computing equipment

into servers [23, 16]. However, sleep states and low power processors are still not widely available in data center type servers. The alternate approach of enabling cluster-level power proportionality has been explored by [14, 17, 18, 19, 20].

[19] deploys a covering subset scheme where they keep one replica of every Hadoop file system block on a small subset of servers. This server subset is always kept powered on to ensure high data availability. This technique, thus, requires modification of the file placement code inside Hadoop. Likewise, [17] modifies the file placement to to partition data into disjoint hot (always powered up) and cold zones. [18] power down entire clusters and run jobs in a batch mode by periodically powering then entire cluster up. This leads to delays for unpredictable arrivals of interactive jobs. [14] adopts this technique only for long jobs, while keeping a small subset of servers on for interactive jobs. This technique requires control over data placement module of the cluster framework, where the algorithm transfers data required for the interactive jobs to the servers that are powered up. [20] provides an online algorithm changing the number of active servers to match the workload, evaluating their technique on production workloads. While Hypnos could be strengthened by the addition of such a technique, the focus of Hypnos was to build a fault-tolerant unobtrusive system which different power management algorithms could leverage.

2.3 Loitering

To harness the idleness for energy savings, Hypnos utilizes the well-established technique of powering down servers when idle. However, if a job request arrives for a powered down server, that request will incur very high latency because the server must be woken up before that request can be served. This takes on the order of minutes. So care must be taken; we cannot put servers to sleep as soon as they become idle without harming performance.

Loiter time is the duration of time a server will remain idle before going it is powered down. Shorter loiter times means servers switch off more frequently, causing more job requests to suffer performance penalties. Longer loiter times keep servers remaining idle longer, decreasing energy savings. The loiter time for servers in a cluster needs to be set only after evaluating these tradeoffs.

2.4 Torque

While the architecture of Hypnos can be applied to different HPC cluster management frameworks, we describe it in the context of the open-source cluster management framework called Torque [10] and a job scheduler called Maui [4]. Torque manages the availability of and requests for compute node resources in a cluster and Maui implements and manages scheduling policies, dynamic priorities, reservations and fair shares of jobs. The Torque server and Maui scheduler resides centrally on the master node of a cluster. The remaining compute servers runs a Torque daemon which executed submitted jobs.

A sample job flow involves a script submitted to Torque specifying constraints. Maui periodically retrieves from Torque a list of potential jobs, available nodes, etc. When desired servers become available, Maui instructs Torque to execute jobs on them. Torque then dispatches the jobs to the compute servers, which then execute the job script. Maui periodically updates its information regarding job execution status. A job spools its output data on to local storage, and at the completion of job execution copies them to the user's NFS directory.

3 Hypnos

Hypnos is a defensive meta-system which unobtrusively provides power proportionality for an HPC cluster. We explain Hypnos in the context of Torque [10], but it can be easily extended to the other HPC job-sharing frameworks such as IBM Load Sharing Facility (LSF) [3] and the Oracle (formerly Sun) Grid Engine (SGE) [5]. The principles guiding the design of Hypnos were

- Unobtrusiveness : Hypnos should not interfere with any existing cluster software or network stack. The administrator should be able to turn the power management feature on or off without having to change any cluster configuration.
- Fault-Tolerance : Hypnos should be able to tolerate various software and network faults which might result due to frequent power-cycling, and should not stall or in any way affect the functioning of the underlying cluster framework or scheduler.
- Extensibility: Hypnos should should be easily adaptable to different HPC cluster frameworks and should allow easy deployment of different power management algorithms.

Brief Overview of Hypnos: Hypnos uses the observation that most existing HPC frameworks (e.g [10, 5]) expose three interfaces which allow unobtrusive power management - (a) An interface to add / remove servers from the cluster framework's purview (b) An interface to expose server state information, and (c) An interface to specify job constraints.

To build an unobtrusive power management system, Hypnos uses the cluster framework's APIs to identify idle servers and shut them down (maintaining a small

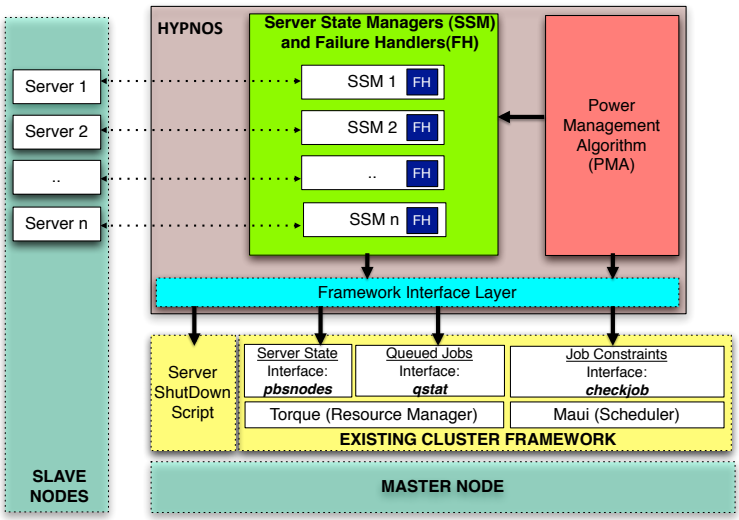


Figure 2: Component diagram of the various components of Hypnos (enclosed in the brown-shaded box). Hypnos lies on top of the Torque deployment on the cluster master node. It uses the *pbsnodes*, *qstat* and *checkjob* interfaces provided by Torque and Maui. For each server, the Server State Manager implements the state diagram shown in Figure 4

spinning-reserve of servers which are powered-up so as to lower response times for new jobs). The cluster state information is obtained through the Framework Interface Layer, which can be re-written for different frameworks thus decoupling the working of Hypnos from the vagaries of the specific cluster frameworks.

However, power-cycling servers may lead to certain software and hardware failures which the power management algorithm may not handle gracefully- e.g a server may not receive a wakeup command, or a server might have a variable wakeup time due to failure to load necessary boot-time networked services (such as NFS, NIS, etc), or the frequent actions of taking a server in and out of the cluster framework's purview might lead to race conditions which may lead the cluster framework to schedule a job on a server which is about to be shut down. Hypnos achieves fault tolerance in such cases by maintaining a state machine for each server[6]. The state machine approach allows Hypnos to reason exactly what failures can occur during each state transition, and how to circumvent them. If any software or hardware fault is inferred on a transition between two states, Hypnos transitions the server instead to a separate *Problematic* state (Figure 4).

The Power Management Algorithm (PMA) implements a wakeup control loop and a shutdown control loop on the servers that are not marked as *Problematic*. It wakes up servers if existing queued jobs cannot be binpacked on to the set of already powered-up free[7] servers (or the set of servers that are currently waking up). The shutdown control loop shuts down servers in case they have been idling for too long provided powering them down does not affect the minimum idle spinning-reserve.

Going back to our original goals, Hypnos thus achieves *unobtrusiveness* by virtue of its meta-system design, where it sits on top of the cluster framework's (Torque's) master node, and only uses information available through the interfaces exposed by it. Hypnos achieves *fault-tolerance* by using a state-machine for each server. Hypnos achieves *extensibility* by decoupling the framework-specific parser, the power-management algorithm and the failure handler modules. The Power Management Algorithm can be optimized in isolation without modifying the other modules, to take into account cluster-specific workload features like its diurnal patterns or its burstiness [26].

Assumptions: We assume, that each server has a health-check script, which is integrated with the cluster framework (such as [11]). The health-check script runs periodically to ensure that all the services required to run jobs are mounted/loaded on the server. If the health-check script fails on a server, the cluster framework is notified which then marks the server as "non-schedule-able". Also, we assume that the master node has the ability to power cycle servers remotely[8]. In the following sections, we will describe complete server shutdowns as

[6]It maintains a Server State Manager Module(SSM) and Failure Handler(FH) for each server

[7]powered-up servers that are not fully utilized

[8]These assumptions are satisfied by most production clusters

Table 2: Hypnos obtains cluster state information using three Torque interfaces. The following table lists the analogous interfaces Hypnos could use on other HPC frameworks - IBM Load Sharing Facility (LFS), and the Sun Grid Engine (SGE)

Torque/Maui	LSF	SGE
pbsnodes	*badmin*	*qmod*
qstat	*bjobs*	*qstat*
checkjob	*bjobs*	*qstat*

a mechanism to eliminate server idle energy consumption. Instead, servers could simply be sent to the most efficient sleep state (S3), and this does not change Hypnos' system design[9]. Also, we do not make use of frequency scaling in our implemented system, although this capability could be easily added to enable further energy savings. The Hypnos system design is shown in Figure 2. In the following subsections, we describe in detail the Hypnos design and the rationale behind each of its modules.

3.1 Framework Interface Layer

The Framework Interface Layer abstracts out the process of obtaining cluster information and actuation from the rest of the Hypnos system, and provides an API which the remaining modules use. This ensures easy adaptability of Hypnos to different cluster frameworks.

To implement power-proportionality in a cluster running a job-sharing framework with a shared file system, one only needs three pieces of information : (a) the status of each node (e.g whether it is active, idle, unable to run jobs or powered off), (b) the list of jobs currently executing or queued up on the cluster, and (c) the placement constraints of each of those jobs. Hypnos obtains this information from the Torque's *pbsnodes*, *qstat* and Maui's *checkjob* interfaces respectively. On the actuation side, a power-proportionality software only needs the ability to manipulate the state of each server. Hypnos achieves this through Torque's *pbsnodes* interface and through the cluster's shutdown script. The description of each of the used interfaces is shown in Figure 3(a). The analogous interfaces on other HPC cluster frameworks is listed in Table 2.

The Framework Interface Layer, with the three methods listed in Figure 3(b), enables Hypnos to decouple the failure handling, sever state management and its power management logic from specific syntax and semantics of Torque and Maui.

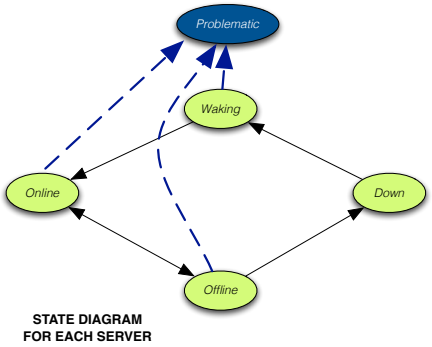


Figure 4: The different states and possible transitions of each server implemented by the Server State Manager

3.2 Server State Manager

Hypnos implements a state machine and a failure handler for each server under its control. The Server State Managers (SSMs) implements the state machine shown in Figure 4. Each server can be in 5 possible states. The transitions can be effected either by the server's SSM or by the common Power Management Algorithm (PMA) module. The objective of the SSM is to decouple the power management logic from failure handling and server state tracking, ensuring a simple and easy-to-improve PMA.

The SSM has a timer associated with each state, which tracks how long a server has been in a particular state. The meaning of a server being in each of the states is elaborated below :[10]

Online: This state signifies that the server is powered up and is either executing jobs or is idle. When a server is *Online*, the timer associated with this state is kept fixed to 0. Thus, the timer associated with this state indicates how long a server has been idle.

Offline: The *Offline* is a transitory state a server goes through before it is shut down. In this state, no job can be scheduled on the server. Torque (as well as other HPC frameworks) has the capability to put a server in a non-schedule-able but powered up state. Hypnos puts a server in this *Offline* state for a sufficient period of time before it is powered down. This state ensures that a race condition between the Power Management Algorithm (PMA) and cluster scheduler is avoided. Such a race condition may take place when a server being shut down by the PMA, lingers in the cluster framework's on-

[9]It is important to note that sleep states are still not available or enabled by default in a lot of the newer servers [16]. Our test cluster also did not have the ability to put servers to S3.

[10]Note that these set of states are different in semantics from the set of states Torque uses internally (which was shown in Figure 3(a))

Interface used	Associated Software	Use
pbsnodes	Torque	1. Get the state of a server. A server can be a. Job-exclusive (Server has no resources left to serve another job) b. Free (Torque can schedule jobs on this node) c. Offline (Torque cannot schedule jobs on this node) d. Down (cannot communicate to Torque client) 2. Change the state of the server from offline to free and vice versa
qstat	Torque	Lists the jobs current running and queued up on the cluster
checkjob	Maui	Provides details (constraints, status and resource requirements) of a submitted job

(a) Torque interfaces used by Hypnos

Module	Methods offered	Use
Server State Manager Objects	getState()	Returns the server's current state.
	getStateTime()	Returns time spent in the current state
	getConfigs()	Returns the server's capabilities
	changeState(*newState*)	Changes the server's state to the new state specified (if possible)
Framework Interface Layer	getServerState (serverid)	Returns the server state reported by Torque
	setServerState (serverid)	Changes server state by the pbsnodes command
	getQueuedJobs()	Returns list of queued and running jobs from the qstat interface
	getJobConstraints (*jobid*)	Returns a job's queue and resource demands from checkjob interface

(b) Methods exposed by each module of Hypnos

Figure 3: Description of the interfaces used between Hypnos modules, and between Hypnos and job sharing framework. The establishment of these interfaces ensures that the Framework specific library or the Power Management Algorithm can be easily modified or extended

line server list for a brief period, leading it to schedule a job on that server.

Down: This state signifies that the server is powered off. A server is not transitioned to the *Down* state until the cluster framework (Torque) reports that the node is unreachable.

Waking : Once the Power Management Algorithm (PMA) wants a server powered up, the SSM wakes the server up[11], and transitions the server to the *Waking* state. This state is an intermediate transition state to account for the time elapsed between servers being powered up to when they become ready to execute jobs. Once the cluster framework's interfaces confirm that the server is online, the state of the server is changed to *Online*.

Problematic: A server is in the problematic state, if the PMA or the SSM detects some kind of failure associated with the server. Failure inference may happen when a server is transitioning from either the *Online, Offline*,or *Waking* states. We consider the reasons for such failures, and elucidate the way to detect and gracefully handle them in the next section.

3.3 Failure Handling

When a server ends up in the *Problematic* state, the server-specific Failure Handler records the state the server last was in, and takes action accordingly.

Last state - *Waking* : For a server to successfully execute jobs, it might have to load multiple different software services over the network (such as the networked file system, DHCP configuration, working directories, etc), the failure of any of which might render it incapable of running jobs. Typically, a health-check script integrated with the cluster framework is run at boot-time to check whether the list of necessary services is running properly [11]. If any of the required services are not loaded properly, Torque is instructed by the health-check script to mark the server as non-schedule-able. This failure is important to handle so as to give the Power Management Algorithm certainty over whether a woken up server can schedule currently queued jobs, or whether another server needs to be woken up as a replacement.

Once a server is powered on, the SSM waits for a configured time interval[12] to check if the node is reported to be schedule-able by Torque. In case, the node continues to be marked as non-schedule-able by Torque, the SSM changes the state of the server to *Problematic*, assumes that the wakeup mechanism might have failed or the necessary software services may not have loaded. It keeps checking the Torque interfaces for any update in the server's status, while periodically repeating the wakeup mechanism just in case the fault was caused by network failure.

Last state - *Offline*: This happens mostly when some running service prevents the server from shutting down, or the shutdown command packets are lost in the network. If this failure was not handled, when the Power Management Algorithm wanted this server woken up to service a new job, the server would not respond to a wakup command because it had never shut down. Bring-

[11]This can be achieved through a `wakeonlan` packet or other mechanisms

[12]5 minutes in our deployment cluster. This was a reasonable amount of time for servers to shut down or wake up in our cluster. This value is taken as a parameter by Hypnos and can be cluster-specific.

ing such a server back online would not be correct, as the server might have unmounted the services necessary to execute jobs. Also, a failure to shut down would lead to a reduction in the energy savings. To tackle it, once the SSM wants a server to be powered down, the SSM sends the shutdown command over the network, and waits for fixed time interval to check if the server is reported as shutdown by the cluster framework (Torque). In case, Torque does not report the server to be shutdown, the shutdown command is resent periodically.

Last state - *Online*: It may happen that the cluster framework's failure detector incorrectly assumes that a powered up server can properly execute jobs. For instance, one failure mode we encountered during deployment was that a few servers ran out of local disk space rendering them unable to run jobs. This resulted in jobs getting queued up, in spite of Torque reporting these servers as schedule-able. The Power Management Algorithm, which depends on Torque's interfaces for server state information, incorrectly inferred that jobs were getting queued despite the presence of free *Online* servers due to some user-specific limits set by the cluster administrator or other Torque fairness mechanisms. To counter such a failure in the *Online* state, the SSM tries to run small dummy jobs periodically on an *Online* server which has been idle for a fixed amount of time, to ensure that they can still execute jobs. The Hypnos Director discounts these jobs while calculating server idle time. A successful execution of the small job implies that the target server is in a correct state and is schedule-able, else it is transferred to the *Problematic* state.

3.4 Power Management Algorithm

The Power Management Algorithm (PMA) module defines the logic behind powering idle servers down and waking them up when new jobs arrive. The PMA in Hypnos is executed periodically (taken as a parameter in Hypnos; every 60 seconds in our deployment) and contains the wakeup and shutdown control logic for each servers. The frequency of execution of the Hypnos Power Management Algorithm can be adjusted to match with the HPC framework's scheduler frequency.

3.4.1 Wakeup Control Loop

There are the two objectives of the PMA in the wakeup control loop. First, the PMA obtains a list of currently queued jobs (obtained from the Framework Interface Layer) and determines which servers to wake up. Hypnos achieves this through a naive bin-packing algorithm, where it goes through the list of free[13] *Online*, *Offline*, *Waking* and *Down* servers (in that order) to look for servers to run the queued jobs. The order is important because, we want to avoid powering up servers for jobs which can be scheduled on those that are currently powered-up or is in the process of being powered up. If a queued job can be packed into an *Online* server[14], Hypnos assumes that job arrived in-between two cluster scheduling iterations, and will be scheduled at the next scheduling iteration. If a job can be run on an *Offline* server, Hypnos brings the server back *Online* because this is faster than to wake up a powered down server. If a job can only be run on a powered down server, or the *Online, Offline* and *Waking* servers have already been filled by the bin-packing algorithm, Hypnos instructs that server's state machine to wake the server up[15].

Second, the PMA maintains a constant headroom of idle servers. A typical cluster may contain servers of different configurations and capabilities. Jobs may have associated constraints that force them to be scheduled on a specific type of server. The PMA automatically groups the server configurations into different classes (each class consists of servers having the exact same configuration), and ensures that there is a headroom of idle spinning servers in each distinct *server class*. This is done to ensure that all servers of an infrequently used *server class* do not get powered down.

3.4.2 Shutdown Control Loop

The shutdown control loop moves *Online* servers that have been idling for more than a specified period of time to the *Offline* state, provided it does not reduce the minimum spinning reserve for its server class. Then, *Offline* servers, which have been spinning idly for more than the configured amount of time and have not accidentally been scheduled jobs by the cluster framework are powered down. Note that the PMA runs the wakeup control loop before the shutdown control loop to ensure that if *Offline* servers can serve queued jobs, they are brought back *Online* and not shutdown.

3.4.3 PMA in the face of Server Failures

The Power Management Algorithm thus designed can circumvent server failures. If a *Waking* server fails to come *Online*, it is flagged as problematic by the SSM. While running the next iteration, the PMA (which disregards *Problematic* servers during its bin-packing phase), will automatically bin-pack the jobs on to a new *Down*

[13] powered-up servers that are not fully utilized

[14] This is a strange occurrence because if a job was schedule-able on an online server, the cluster scheduler should already have scheduled it

[15] A *Down* server is only woken up if a sufficient time has passed since it was transitioned to the *Down* state. This time period ensures that the server shuts down safely, and can be configured by changing a parameter in Hypnos

server, which will then be woken up. This ensures that enough servers are eventually woken up to serve a queued job. In case an *Online* server cannot run the SSM's dummy jobs and is transitioned to the *Problematic* state, the PMA in its next iteration will wake up a *Down* server to ensure a constant headroom of *Online* servers. Thus, the headroom will always consists only of non-problematic servers which can successfully run jobs. The PMA assumes that a queued job exceeded cluster fairness/user-specific limits when the job can run on a free *Online* server but is not being scheduled by the cluster scheduler. In such a scenario, the PMA will not power up an extra server to service the queued job.

The PMA could be augmented with predictive models of the cluster-specific workload. For example, the PMA could take into account the burstiness and the diurnal nature of the workload and power up idle servers before hand. This forms part of our future work, where Hypnos would have a module to automatically analyze existing traces (using techniques similar to [26]) on the cluster where it is being deployed, in order to anticipate server power ups.

4 Implementation and Results

We deployed Hypnos on an academic cluster in Berkeley. The cluster is used by about 40 Artificial Intelligence, Machine Learning and Computer Vision graduate students.

Server configurations: We deployed Hypnos on 57 servers of a cluster for 21 days. 51 of the servers were Dell PowerEdge 1850 servers having 2 cores running at 3.0GHz, 3GB RAM, consuming 192 W at idle and 292W at peak utilization. The remaining 6 servers were Dell PowerEdge 1950 servers with 8 cores at 2.3GHz, 16GB RAM, with an idle power of 253W and a peak power of 387W. The servers were automatically grouped into five classes based on the queues they belonged to and their hardware configurations, according to Torque's node configuration file.

Hypnos configuration: Hypnos is written in Python and the entire system is 1349 lines long. The Power Management Algorithm module is only 346 lines. Hypnos is deployed on the cluster head node, where the Torque master resides. A 285-line health-check script was already deployed on each compute server. Servers were powered down remotely using the `shutdown` command in a bash script which had administrator privileges on all servers. Servers were powered up using the `wakeonlan` command.

4.1 Characteristics of the test cluster

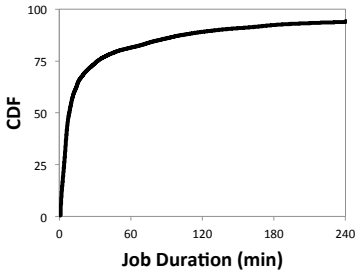
In order to optimize the parameters of Hypnos for the most possible energy savings and to demonstrate that our testbed was a representative academic cluster, we analyzed a 68-day trace taken from the cluster when power proportionality was not deployed. During this period the cluster had executed about 169,000 jobs, and the average cluster utilization was 45%.

Figure 5(a) shows the CDF of the job durations submitted to the cluster during this period of time. Approximately 50% of the jobs take less than 10 minutes duration. This is probably due to the fact that users of the cluster have a debug cycle, where they run their jobs on small portions of data in order to check correctness of their code before running it on the full data set. This graph shows the need for a spinning reserve of idle servers (headroom). A headroom provides fast turnaround times for small jobs. Without the headroom, small jobs (of less than 10 minutes duration) may have to wait for a powered-down server to spin up (which may take 4-5 minutes), resulting in users' debug cycle being extended by at least 40-50%. There is a tradeoff in choosing the number of servers to keep as headroom — a large headroom results in lower energy savings (because fewer idle servers will be powered down), while a smaller headroom impacts completion times for small/debug jobs. In our experiments we choose a point in this tradeoff, keeping a headroom of 3 idle servers *each server class*.

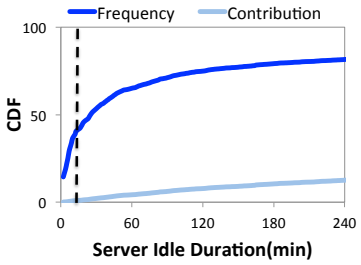
Figure 5(b) shows the characteristics of the idle durations of the various servers in the cluster. The graph shows the CDF of idle durations, as well as the CDF of an idle period's contribution to the sum of all idle durations during the period. 40% of the idle periods are less than 10 minutes in duration, following which the CDF curve begins to flatten out. The contribution of the idle durations of less than 10 minutes to the whole cluster idle is about 0.7%. This indicates that large idle durations are responsible for most idle-energy wastage. The conclusion is that some servers in the cluster run jobs very infrequently, and just powering them down would yield large energy savings without very little impact on job performance. We choose the total loiter time[16] of an idle server to be 10 minutes because it allows us to be moderately aggressive in shutting down servers, and yet harness most of the idleness in the cluster for energy savings.

Figure 5(c) shows the impact of various loiter times on energy savings and the amount of time a job would have to spend in the queue waiting for a powered down server to spin up. Too small a loiter time would have resulted in lot of energy savings, but increased the job response

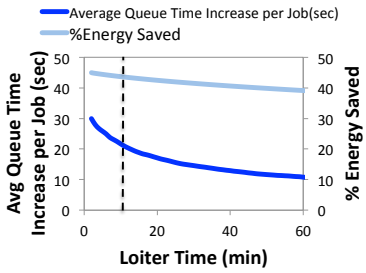
[16]The time before a powered up idle server is shut down



(a) Cumulative Distribution Function of Job durations on our test cluster



(b) Cumulative Distribution Function of the idle periods in the cluster by frequency, and contribution to the total idle period of the entire cluster



(c) Impact on energy savings and job response time for various loiter times in the cluster

Figure 5: Characteristics of the test cluster before Hypnos was deployed

times. We see that a 10 minute loiter time would increase average job queue times by only 20 seconds, while saving almost 40% of the energy. Note that increasing loiter times does not have a marked response on the amount of energy saved. As explained in the previous paragraph, the servers contributing most to the total cluster idleness run jobs rarely. So, even if we increase the loiter time these servers will eventually get powered down, resulting in large energy savings.

During our experiments, we divide the 10-minute loiter time into 7min loiter in the *Online* state and a 3min loiter in the *Offline* state. The *Offline* state loitering is done to ensure that Torque does not mistakenly schedule jobs on a server soon-to-be-powered-down due a race condition between Torque and Hypnos.

4.2 Results from deployment

Hypnos was deployed without any changes to the existing Torque deployment. The energy reduction summary and performance impact from the Hypnos deployment is shown in Table 3. During our 21-day deployment, the cluster utilization rose marginally from the previous value to 46%.

Figure 6(a) shows the variability of the power profile of the cluster after Hypnos was deployed. Hypnos was able to save 36% energy compared to a scenario if it had not been deployed(The ideal possible energy savings was 37.5%). Figure 6(b) shows that the power profile closely matched cluster utilization, showing that Hypnos was able to switch the idle servers off effectively. Also, the number of servers switched on at any point of time was only slightly more than the number of active servers (servers running jobs).This demonstrates Hypos' reliability in powering servers up and down, and maintaining server headroom.

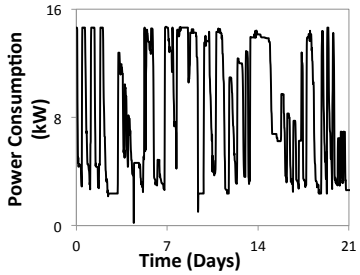
Figure 6(c) shows the impact of our parameter choices for loiter time and headroom on job performance. The CDF of the percentage of time a job spent in the queue time as compared to its execution time is shown separately for relatively small jobs (less than 30 min duration) and larger jobs (greater than 30 min duration). Almost 50% of all jobs faced less no increase in execution time. This is because of the headroom in each server class, which was able to serve jobs as soon as they were submitted. Almost 80% of the larger jobs had a less than 10% increase in their execution time because of encountering powered down servers. The remaining 50% of smaller jobs had a larger percentage increase in their job duration because the time for a server to wake up is large compared to the job's execution time. Some large as well as small jobs showed more than a 50% increase in their execution times. This was due to certain users submitting a large number of jobs at once, resulting in the cluster getting fully utilized, resulting in large queueing times.

If the test cluster had consisted of newer 2010 HP Proliant DL385 servers (instead of the older PowerEdge 1850 and 1950 servers) with the same workload, Hypnos would have saved 23% energy (with the optimal being 26%). Also, such a new cluster equipped with the latest version of Linux (which brings boot times to 2 minutes), would have seen an average job delay of 9s (as opposed to 22s on our test cluster).

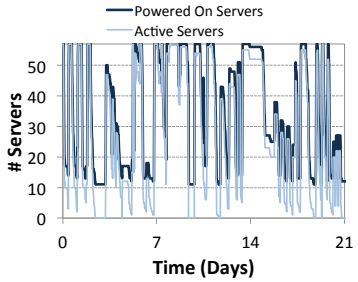
4.3 Failure handling

Since Hypnos was deployed on a server that was in constant use, Hypnos had to deal with several software failures arising out of frequent server power cycling. This section shows certain failure modes that Hypnos was effectively able to deal with.

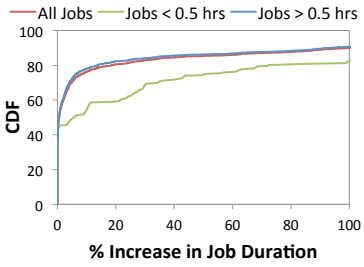
Figure 7(a) shows the variable server wakeup times in the test cluster. At boot time, each server in our test



(a) Power consumption of cluster under Hypnos

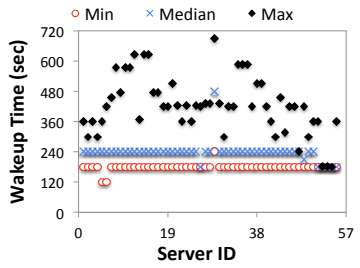


(b) The number of active servers (that are running jobs) and the number of powered up servers during the course of our deployment.

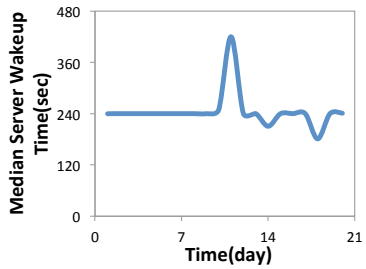


(c) Cumulative Distribution Function of a job's queue time as compared to it execution time (%Increase in Job duration)

Figure 6: Power consumption and performance impact under Hypnos' operation



(a) Variation in wakeup time by server ID



(b) Daily variation is server wakeup time

Failed Service	Number of Occurances
Network Information Server	777
Network File System	773
Work Directory Server	4

(c) Frequency of service mount failures at reboot

Figure 7: Reliability and Fault-tolerance achieved by Hypnos duirng the 21-day run

Metric	Value
Avg. Utilization	46%
Avg. Energy Savings	36%
Ideal Energy Savings Possible	37.5%
Number of submitted Jobs	3651
Total number of Server Reboots	1094

Table 3: Usage statistics from the Hypnos deployment over 21 days on 57 servers

cluster has to contact the Network File System (NFS) server, the Network Information Service (NIS) server, and a server to load separate work directories before it is ready to run a job. Depending on the load on the NIS and NFS and work directory servers, the wakeup time of the cluster nodes may vary. While the median server wakeup time for most servers was 240 seconds, a server reboot might take as long as 720 seconds due to failure in mounting the NFS file system, working directories, or contacting the NIS server. Hypnos ensures fault tolerance by enforcing a restart timeout of 5 minutes, before which it powers up another similar server.

Also, during the course of the deployment, the building network developed some transient network failures. This resulted in the NFS and NIS servers responding too slowly, causing every server reboot to take much longer (Figure 7(b)). The median server wakeup time across all servers jumped from 240 seconds to almost 420 seconds on that day. Hypnos effectively circumvented this temporary drop in network performance through its failure handlers. Figure 7(c) shows the number of times required software services failed on servers in the cluster during the deployment. These software service failures resulted in a delay in server wakeup, or rendered a server unable to run jobs. Again, Hypnos was able to circumvent these failures by powering up other servers in its place.

5 Conclusion and Discussion

In this paper, we have demonstrated Hypnos - a power-proportionality meta-system, which is unobtrusive, reliable and flexible in its design. We argued that a meta-system approach is more general (applicable to differ-

ent resource managers), cost-efficient (in terms of code maintenance and ease of deployment) and felixble (allows the resource management software to update its code base without considering power proportionality). Although, we deployed and tested Hypnos on a Torque cluster, the design decisions and interfaces used have analogues in other HPC job-sharing frameworks such as LFS and SGE. The main aim behind developing Hypnos was to provide an open-source solution to cut down on the idle energy consumed in under-utilized clusters. We report results from Hypnos over a 21-day period, where it was able to save 36% of the energy without succumbing to hardware or software faults.

There are various ways to improve on the Hypnos architecture and algorithms. A more optimized power management algorithm could be written which considers the server wakeup actuation as stochastic. It can keep track of the average wakeup times of each server, and power on servers with a low wakeup time. Also, in instances where past data shows that a server wakeup time is unreliable (has a high variance), it could power on more servers than required in order to serve the queued jobs. Second, the power management algorithm in Hypnos, though effective, can be further augmented with predictive techniques such as [20, 26]. We believe that these power management algorithms can be easily incorporated into the Hypnos architecture against the server state-machine abstraction.

6 Acknowledgments

We would like to thank Jeff Anderson-Lee, Albert Goto and Andrew Krioukov for help in setting up the experiments. This work was funded in part by NSF Grants CPS-0932209 and CPS- 0931843.

References

[1] Green Computing powered by Moab. http://www.clusterresources.com/solutions/green-computing.php.

[2] Google data center PUE performance. http://www.google.com/about/datacenters/efficiency/internal/.

[3] IBM Platform LSF. http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html.

[4] Maui Scheduler. http://www.adaptivecomputing.com/resources/docs/maui/pbsintegration.php.

[5] Oracle Grid Engine. http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html.

[6] RENERGY. http://xcat.sourceforge.net/man1/renergy.1.html.

[7] ROCKS-CLUSTERS. https://code.google.com/p/rocks-solid/wiki.

[8] ROCKS-SOLID. https://code.google.com/p/rocks-solid/wiki.

[9] The Grid Workloads Archive. http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Main.Home.

[10] Torque Resource Manager. http://www.adaptivecomputing.com/products/open-source/torque/.

[11] Warewulf Node Health Check. http://warewulf.lbl.gov/trac/wiki/Node\%20Health\%20Check.

[12] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 1–14. ACM, 2009.

[13] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec. 2007.

[14] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *EuroSys*, pages 43–56, 2012.

[15] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[16] A. Gandhi, M. Harchol-Balter, and M. A. Kozuch. The case for sleep states in servers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 2:1–2:5, New York, NY, USA, 2011. ACM.

[17] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In *Proceedings of the 2010 IEEE Second International Conference on Cloud*

Computing Technology and Science, CLOUDCOM '10, pages 274–287, Washington, DC, USA, 2010. IEEE Computer Society.

[18] W. Lang and J. M. Patel. Energy management for mapreduce clusters. *Proc. VLDB Endow.*, 3(1-2):129–139, Sept. 2010.

[19] J. Leverich and C. Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, Mar. 2010.

[20] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 1098–1106. IEEE, 2011.

[21] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 175–186. ACM, 2012.

[22] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 205–216, New York, NY, USA, 2009. ACM.

[23] D. Meisner and T. F. Wenisch. Dreamweaver: architectural support for deep sleep. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 313–324. ACM, 2012.

[24] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 343–356, New York, NY, USA, 2011. ACM.

[25] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., second edition, 1996.

[26] K. Wang, M. Lin, F. Ciucu, A. Wierman, and C. Lin. Characterizing the impact of the workload on the value of dynamic resizing in data centers. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 405–406. ACM, 2012.