Implications of High Energy Proportional Servers on Cluster-wide Energy Proportionality

Daniel Wong Murali Annavaram
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
{wongdani,annavara}@usc.edu

Abstract

Cluster-level packing techniques have long been used to improve the energy proportionality of server clusters by masking the poor energy proportionality of individual servers. With the emergence of high energy proportional servers, we revisit whether cluster-level packing techniques are still the most effective way to achieve high cluster-wide energy proportionality. Our findings indicate that cluster-level packing techniques can eventually limit cluster-wide energy proportionality and it may be more beneficial to depend solely on server-level low power techniques. Server-level low power techniques generally require a high latency slack to be effective due to diminishing idle periods as server core count increases. In order for server-level low power techniques to be a viable alternative, the latency slack required for these techniques must be lowered. We found that server-level active low power modes offer the lowest latency slack, independent of server core count, and propose low power mode switching policies to meet the best-case latency slack under realistic conditions. By overcoming these major issues, we show that server-level low power modes can be a viable alternative to cluster-level packing techniques in providing high cluster-wide energy proportionality.

1. Introduction

Low energy proportionality of datacenter servers has been a major obstacle for achieving datacenter energy efficiency. This led to many research solutions that provide low power operating modes at various levels of the datacenter hierarchy, in order to improve the overall energy proportionality. Low power modes can be classified into three granularities: component-level, server-level, and cluster-level.

Component-level low power modes target individual components within a server, such as DVFS for processors, MemScale [7] and MemBlaze [14] for memory, and disk spin down for hard drives. These techniques can take advantage of very fine idle periods that are less than a second long. Historically, the processor has accounted for the majority of server power consumption, but recent work has shown that no single component dominates server power consumption [9, 12, 24]. In order to achieve greater energy savings, there needs to be coordinated low power modes across all components [16].

Server-level low power modes, such as PowerNap, Barely-alive servers and KnightShift [2, 15, 26], achieve greater energy efficiency by coordinating power consumption across multiple components within a server. Server-level low power modes can be further classified into either an *active* or *inactive* low power mode. Inactive low power modes do not perform any work in a low power state, while active low power modes can still perform useful work, albeit at a reduced performance level. Commercially available inactive low power modes, such as server shutdown and sleep/hibernate, require large idle periods in the order of minutes to become effective. To combat the need for large idle periods, recent techniques such as PowerNap [15], were proposed to take advantage of sub-second idle periods. Server-level inactive low power modes depend on the presence of idle cycles, but with the emergence of multicore processors and increasing core count within servers, idle periods are becoming shorter and are increasingly rare [17]. Recognizing this concern, prior work [8, 17] proposed idleness scheduling techniques in order to artificially create idle periods at the cost of increased response time.

Active low power modes can still perform work in a low power state. Barely-alive servers [2] and Somniloquy [1] only handle I/O requests in their low power state. KnightShift [26] can handle general computation and takes advantage of *low utilization periods*. Unfortunately, these approaches also demand relatively high latency penalties in order to achieve reasonable energy savings.

An orthogonal approach to improve energy proportionality is to use cluster-level low power techniques. Traditionally, clusters are managed with the goal of improving response time through uniform load balancing, where the workload is uniformly distributed to all servers in the cluster. This technique is simple, but can be energy inefficient, especially during low utilization periods, which keeps all servers on. Recognizing this concern, researchers have pro-

posed many cluster-level power saving techniques. Power and cooling management techniques [9, 19–21, 23] have been explored across the server, rack, and datacenter granularity to reduce power consumption and heat, and maximize capacity. Workload consolidation techniques [6, 22] have the goal of migrating workloads to improving the utilization of datacenters to reduce the number of servers required, resulting in improved TCO. Dynamic capacity management [10, 25] work towards the goal of minimizing the number of servers needed for a given workload utilization in order to turn off a subset of servers using *Packing* scheduling algorithms. The challenge with Packing schemes is in maintaining the optimal number of servers to keep on in order to meet QoS levels in the face of sometimes unpredictable and bursty incoming work requests, while also minimizing energy consumption. The most recent work to address this problem is AutoScale [10], which showed that conservatively turning off servers with the proper threshold can meet QoS levels while minimizing energy usage.

In this work, we limit our evaluation to cluster-level packing algorithms. Cluster-level packing techniques were originally developed to mask the effect of the poor energy proportionality of individual servers. Over the past few years, however, server energy proportionality has improved drastically [26]. Now with the emergence of high energy proportional servers, we feel it is warranted that we revisit if cluster-level packing techniques are still the best approach to achieving high cluster energy efficiency. The question we would like to answer is, can server-level energy proportionality improvements alone translate into *cluster-wide energy proportionality* (the observed energy proportionality of the whole cluster), or do we still need cluster-level proportionality approaches?

This paper tackles this question and several critical related issues to make the following contributions:

- (Section 2) We extended the energy proportionality model proposed in [26] to measure energy proportionality at the cluster level. We then explored the effect of cluster-level and server-level techniques on cluster-wide energy proportionality. We found that cluster-level packing techniques effectively mask the server's energy proportionality profile, achieving good cluster-wide energy proportionality. Furthermore, we found that cluster-level packing techniques may now hinder cluster-wide energy proportionality with the emergence of high energy proportional servers. As server energy proportionality improves, we conclude that it may be beneficial to shift away from cluster-level packing techniques and rely solely on server-level techniques.

- We found that running a cluster without any cluster-level packing techniques can actually achieve higher cluster-wide energy proportionality than with cluster-level packing techniques. Furthermore, removing cluster-level packing techniques exposes the underlying server's energy proportionality profile, enabling server-level low power modes to now have an effect on cluster-wide energy proportionality to further improve cluster-wide energy efficiency.

- (Section 3) We explored server-level low power modes to understand how the efficiency of these low power modes scale with increasing core count. We performed a detail power consumption analysis of how various server-level low power modes perform under server multicore scaling and found that active low power modes consistently outperform inactive low power techniques using idleness scheduling techniques, while requiring significantly less latency impact in order to be effective.

- (Section 4) In order to meet the best-case latency slack required for server-level low power modes to be efficient, we explore the causes of high latency in server-level active low power modes. We propose various mode switching policies to overcome the high latency currently experienced with server-level active low power modes.

2. Cluster-wide Energy Proportionality

In this section, we first extend the energy proportionality model presented in [26] to measure cluster-wide energy proportionality. We will use our extended model to explain the reasoning behind the effectiveness of prior cluster-level packing techniques. We will then reason about how improved energy proportionality at the server-level is impacting cluster-wide energy proportionality. Specifically, we make the following observations. 1) Cluster-level packing techniques are highly effective at masking individual server's energy proportionality. 2) On the flip side, significant improvement in server energy proportionality seen in the past few years do not translate into cluster-level energy efficient gains due to the masking effect. 3) To take advantage of improved server energy proportionality it may be more favorable to forego cluster-level packing techniques entirely. Foregoing cluster-level packing techniques enable energy improvements by server-level low power techniques to translate to cluster-wide energy improvements.

2.1. Measuring Energy Proportionality

Figure 1 shows an illustrative utilization vs peak power usage curve, also known as the *energy proportionality curve*, as defined in [26]. The top line represents the *actual* server's energy proportionality curve, while the bottom line represents the *ideal* energy proportionality curve. Using this figure, energy proportionality (EP) is defined as:

$$\mathrm{EP} = 1 - \frac{\mathrm{Area}_{actual} - \mathrm{Area}_{ideal}}{\mathrm{Area}_{ideal}} \quad (1)$$

where Area_{actual} and Area_{ideal} is the area under the server's actual and ideal EP curve, respectively.

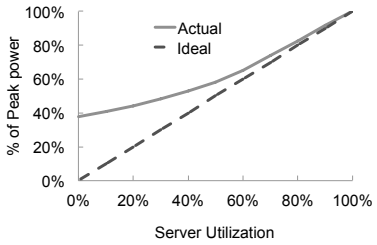


Figure 1: Energy proportionality curve [26]

2.1.1 Measuring Idealized Best-Case Cluster-wide EP

We first describe how we measure cluster-wide energy proportionality using an idealized Packing algorithm. Using the Packing approach, the best-case cluster-wide energy proportionality is dependent on the number of servers used in the cluster. Under an ideal scenario, cluster-level packing technique can always perfectly provision the right number of servers to meet the current utilization demand. Then the cluster-wide energy proportionality curve would resemble steps as shown in figure 3. In these illustrative figures, the x-axis is the cluster-level utilization and the y-axis is the cluster-wide power consumption. Figure 3a shows the best-case cluster-wide energy proportionality with a cluster consisting of 10 servers. The shape of each step resembles the shape of each individual server's energy proportionality curve. In this example illustration, each step represents the energy proportionality curve of a server with a relatively low energy proportionality of 0.24. The best case Packing approach represents the case where the exact number of servers are on for the current utilization level, then the next step occurs when another server needs to be turned on to handle the additional load of a higher utilization. As the utilization increases, it will only load the most recently woken up server until another server needs to be awoken. Similarly, in the base case Packing approach, if the utilization decreases then a server can be instantaneously put to low power mode. Using this best case Packing approach, we compute the best case cluster-wide energy proportionality (which is represented as BestEP) using Equation 1, which is also shown on the top left corners of Figure 3a.

As the number of servers in a cluster increases, these steps become smaller until the point where the cluster-wide energy proportionality curve resembles the ideal energy proportionality curve as shown in figure 3b. When increasing the cluster size from 10 servers to 100 servers, the best achievable cluster-wide energy proportionality approaches 1, improving from 0.92 to 0.98, even though each server suffers from low energy proportionality of 0.24.

In the absence of cluster-level packing techniques, requests may be routed uniformly across all servers to balance the load. In this case, each server's utilization should track the cluster's utilization almost perfectly. When using Uniform load balancing approach, the best-case cluster-wide energy proportionality curve would simply be that of the underlying server's energy proportionality curve.

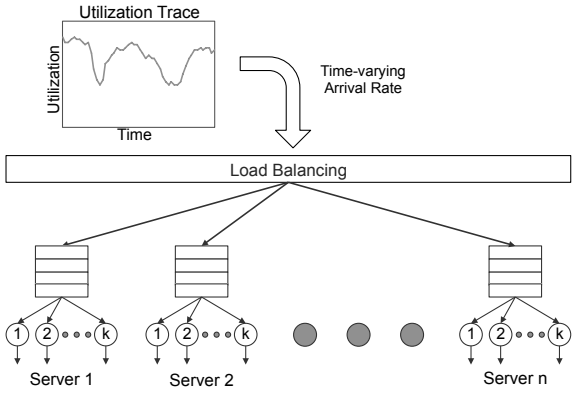


Figure 2: Trace-driven Queueing Model-based Simulation Methodology [26]

2.1.2 Measuring Actual Cluster-wide EP

In real clusters it is not possible to achieve the best case scenario where servers can sleep and wakeup instantaneously. Cluster-wide utilization vs power measurements are relatively noisy due to the fact that at any given cluster utilization, there could be a differing number of servers that are on. For example, consider two scenarios. In both scenarios the cluster utilization during a given time epoch is 10%. In scenario#1 a large number of servers (much more than what is needed for 10% cluster utilization) were already on during the previous epoch. The cluster power consumption can be high even at 10% utilization since it is not possible to instantaneously pack workloads and turn on/off unneeded servers for the current epoch. Consider scenario#2 where there are only just enough servers turned on to meet 10% cluster utilization demand even during the previous epoch. Then the power consumption in the current epoch with 10% utilization will be lower in scenario#2 than in scenario#1.

In order to enable the measurement of cluster-wide energy proportionality, we first take the average power at each measured utilization and then find a 3rd degree polynomial best fit curve to create an average power curve. We then use this curve as the actual energy proportionality curve to calculate cluster-wide energy proportionality using equation 1.

2.2. Evaluation Methodology

To evaluate the implication of various cluster-level and server-level techniques on cluster-wide energy proportionality, we implemented a trace-driven queueing model-based simulator shown in figure 2. We model a cluster with 20 servers, where each server is modeled as a G/G/k queue.

We found that using a cluster size of 20 servers gave us the required resolution to measure cluster-wide proportionality, while minimizing the amount of simulation time required. Using a larger cluster would result in higher power measurement resolution, but would not change our observations.

Due to the absence of individual service request times in our utilization traces, we use a verified G/G/k queueing model methodology based on the concept of capability [26]. In this model, k represents the capability of a server. Each server has a capability of $k = 100$ (since each server can have up to 100% utilization). The queueing model is driven by real-world utilization traces containing minute-granularity utilization measurements from various clusters taken over a 9-day period from an institutional datacenter. The workload traces are detailed in section 4.1. Each entry in the trace shows the cluster utilization at that given minute epoch. The utilization traces are used as a proxy to derive a time-varying arrival rate. For instance, if the cluster has 10% utilization during a given epoch then 20 servers cumulatively will receive 200 jobs. If the cluster utilization drops to 1% then during that epoch only 20 jobs arrive. Because these datacenter traces do not report actual workload response time, we assume the service rate is exponential with a mean of one second and relative performance impact (99th percentile latency) is reported. This still enables us to obtain latency impact by comparing relative performance.

In our simulator, we evaluated three different server categories; LowEP, MidEP and HighEP servers. The low EP server, with an EP of 0.24, is based on a Supermicro server with dual 2.13GHz 4-core Intel Xeon L5630 and consumes 156W at idle and 205W at full load. The medium EP server is an HP ProLiant DL360 G7 with an EP of 0.73. The high EP server is a Huawei XH320 with an EP of 1.05. We used the energy proportionality curve reported in [26] (for LowEP) and SPECpower [27] (for MidEP and HighEP) as our power model used in the simulator. To capture transition energy penalties in the model, an empirically measured constant power of 167W is conservatively added during the entire transition period. To measure cluster-wide power at minute granularity, we simply aggregate each server's power consumption during simulation runtime.

To explore the impact of cluster-level packing techniques, we implemented a state-of-the-art dynamic capacity management algorithm proposed in Autoscale [10]. Each server is configured with the same settings as in [10], where the servers conservatively turn off after 120 seconds of idleness and has server wakeup time of 260 seconds. Through empirical experiments, it was determined that the packing factor for our servers is 97 in our simulation framework. The packing factor is server dependent and indicates the maximum number of jobs that the server can handle and still meet the target 99th percentile latency. The Autoscale load balancing algorithm assigns the incoming requests to individual servers. For instance, if there are only two servers currently turned on and the new utilization trace record has 6% cluster utilization then it translates to 120 jobs that will be submitted to these two servers. The Packing algorithm first submits 97 requests to the first server and then assigns the remaining 23 to the next server. If the arrival rate exceeds the total capability of all active servers then a new server is turned on with a wakeup latency and the remaining overflowed requests will join the shortest queue.

To understand how well Packing load balancing improves cluster-wide energy consumption, we also implemented a basic Uniform load balancer as an alternative to Autoscale. The load balancer simply distributes work equally to all servers and all servers in the cluster are always on.

In addition, we also explored the effect of a server-level active low power mode, KnightShift, on cluster-wide energy proportionality. KnightShift is a heterogeneous server which contains both a high-performance high power Primary node, tightly coupled with a low-performance low power Knight node. During low utilization periods, the high power primary node will shift work to the low power Knight node, and vice versa. The KnightShift server is configured as in [26] with a 15% capable Knight. The Knight is capable of handling 15% of the work of the primary server. In the queueing model, when in Knight mode, k is set to 15. The Knight consumes 15W at idle and 16.7W at full load with wakeup transition time of 20 seconds.

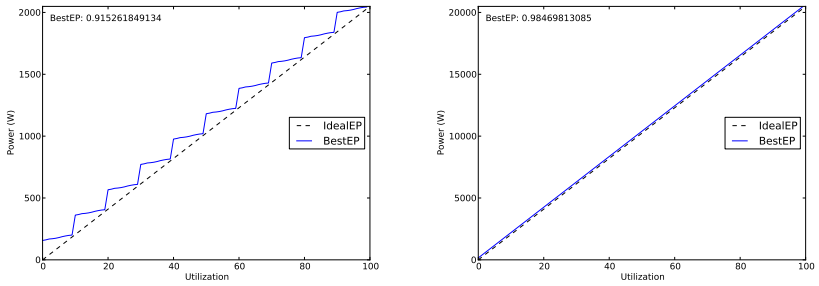
2.3. Revisiting Effectiveness of Cluster-level Packing Techniques

Figure 4 shows the results of our study. Each plot shows the linear ideal energy proportionality curve (IdealEP), the best-case energy proportionality curve for that scenario (BestEP), and the actual fitted energy proportionality curve (ActualEP) derived from the measured (raw) utilization vs power data collected during runtime. In this section, we only ran a 1-day period of the utilization traces as we found this is sufficient to obtain the cluster-wide energy proportionality curves.

Note that for figures 4(a)-(c) the BestEP curve is obtained by assuming the best case Packing algorithm as described earlier in the context of figure 3 with no wakeup and sleep latencies. For figures 4(d)-(f) that does not use Packing, the cluster-wide BestEP is the energy proportionality curve of the underlying servers since all servers are always turned on and each server has the same utilization.

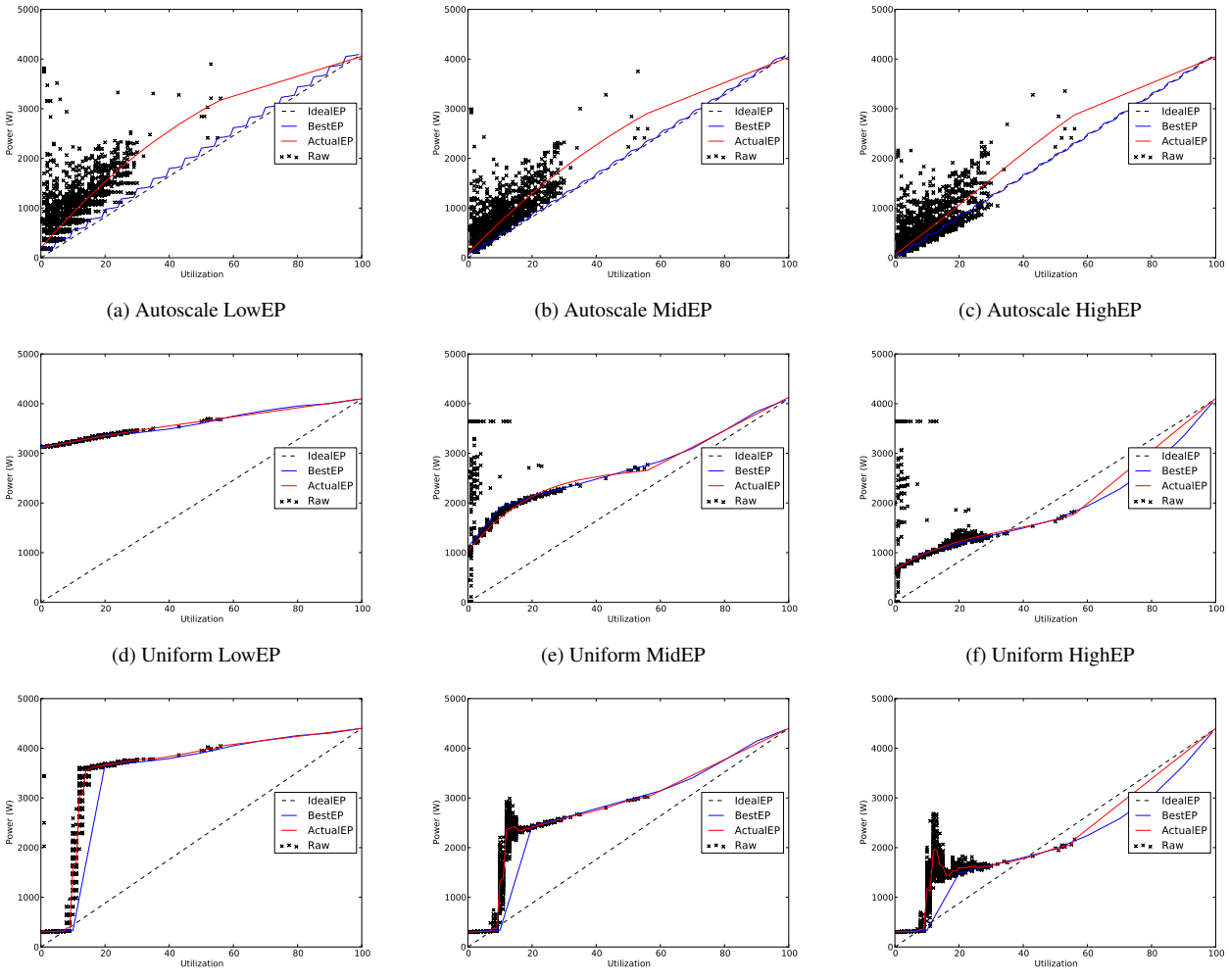
Observation 1: Cluster-level packing techniques are highly effective at masking server EP

Using a Uniform load balancing technique where jobs are sent equally to all servers (lack of cluster-level low power technique) would result in the cluster-wide energy proportionality exhibiting an EP curve resembling that of the server's EP curve as shown in figure 4(d)-(f). Hence,



(a) Cluster-wide EP with 10 servers (b) Cluster-wide EP with 100 servers

Figure 3: Best-case cluster-wide energy proportionality curve using Packing load balancing. As cluster size increases, the best-case cluster-wide energy proportionality approaches ideal energy proportionality.



(a) Autoscale LowEP (b) Autoscale MidEP (c) Autoscale HighEP

(d) Uniform LowEP (e) Uniform MidEP (f) Uniform HighEP

(g) Uniform KnightShift LowEP (h) Uniform KnightShift MidEP (i) Uniform KnightShift HighEP

Figure 4: Cluster-wide Energy Proportionality using (a,b,c) Packing load balancing (Autoscale), (d,e,f) Uniform load balancing, and (g,h,i) Server-level active low power technique (KnightShift). We explored servers with low (0.24), mid (0.73), and high (1.05) energy proportionality. As individual server energy proportionality improves, it may be favorable to forego Packing load balancing in favor of Uniform load balancing. Despite high individual server energy proportionality, there still exist significant opportunity for KnightShift at low utilization periods.

the actual cluster-wide energy proportionality is 0.24 in figure 4d, 0.73 in figure 4e, and 1.05 in figure 4d)

Figure 4a shows how cluster-wide energy proportionality changes using Autoscale. Using low energy proportional servers (EP = 0.24) as a base, the cluster-wide energy proportionality improved to 0.69, compared to just 0.24 with Uniform load balancing with no cluster-level packing (figure 4d). This demonstrates the effectiveness of cluster-level techniques, like Autoscale, at masking the individual servers' low energy proportionality. As individual server energy proportionality improves, the cluster-wide energy proportionality also improves, but the effectiveness is reduced. For instance, MidEP servers in figure 4b achieve a cluster-wide energy proportionality of 0.79 with Autoscale, while the Uniform load balancer achieves cluster-wide energy proportionality of 0.73 (an improvement of only 0.06).

Observation 2: With improving server energy proportionality, it may be more favorable to forego cluster-level packing techniques entirely.

As server energy proportionality improvements continue, servers are beginning to exhibit super-energy proportional (EP $>$ 1.0) behaviors. When Packing is applied to HighEP servers, it does not improve the overall cluster-wide energy proportionality. Having servers with EP of 0.73 (figure 4b) vs 1.05 (figure 4c) had very little effect at the cluster level, where cluster-level energy proportionality improved from 0.79 to just 0.82. *With high energy proportional servers appearing, it may be more desirable, from an energy proportional standpoint, to uniformly load balance work and depend on server-level low power techniques to further improve energy proportionality.*

In the past, it was clear that cluster-wide energy proportional using cluster-level packing technique is better than the individual server's energy proportionality (figure 4a vs 4d and 4b vs 4e). But we may now have reached a turning point where servers may now offer more energy proportionality than cluster-level packing techniques can achieve (figure 4c vs 4f).

Observation 3: Foregoing cluster-level packing techniques enable energy improvements by server-level low power techniques to translate to cluster-wide energy improvements.

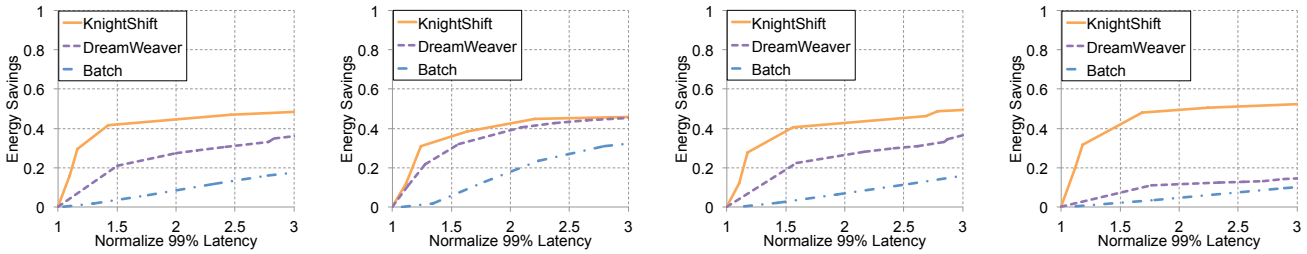
Note that when individual server energy proportionality improves, the majority of the energy proportionality improvement occurs at mid to high utilization levels (figures 4e, 4f). Servers are still the most energy inefficient at low utilization levels. A way to improve energy efficiency at low utilization levels is by improving the dynamic range of servers, but previous work has showed that this is not likely to occur [26]. By switching to a Uniform load balancing scheme and exposing the underlying server's energy proportionality curve, it is possible to apply server-level low power modes to further improve server energy efficiency at low utilization levels. Previously, cluster-level techniques would mask the effect of these server-level techniques, rendering them ineffective. A switch to uniform load balancing can enable server-level low power modes to have more overall benefits by exposing the underlying server's energy proportionality curve.

Figure 4(g)-(i) shows the cluster-wide energy proportionality curve using KnightShift servers with Uniform load balancing. Due to the jump in power due to the Knight/Primary server, polynomial curve fitting does not fit well to create the ActualEP curve. Instead, for KnightShift, the ActualEP curve reported is simply the average power curve. While KnightShift is able to lower individual server's energy proportionality, which is reflected in the cluster-wide energy proportionality, it does not provide better cluster-wide energy proportionality than cluster-level packing techniques (figure 4a and 4b) when the baseline server has relative low energy proportionality. This was also the case with Uniform load balancing. Only with higher server energy proportionality does KnightShift provide better cluster-wide energy proportionality than cluster-level packing techniques. Both Uniform load balancing and KnightShift with high energy proportional servers exhibit EP near 1, outperforming cluster-level packing technique which has EP of 0.82.

Although the energy proportionality may be similar when comparing Uniform load balancing with high energy proportional servers and KnightShift (figure 4f vs 4i), KnightShift still offers significant energy efficiency improvements, especially at low utilization regions. By using a low power Knight mode, KnightShift enables the server to use only a fraction of the primary server's power at low utilization periods. This resulted in the average power usage falling from 890W to 505W using KnightShift.

2.4. Challenges facing adoption of server-level low power modes

In order for system-level low power modes to become more widely adopted, issues relating to its practicality must be resolved. All server-level low power modes, such as PowerNap [15] and Dreamweaver [17], Barely-alive servers [2], Somniloquy [1] and KnightShift [26] trade off latency to improve energy efficiency. Each technique requires varying amounts of *latency slack*, the latency impact required before they become effective. Previous work [17] has identified that server-level low power modes require increasing latency slack as the number of processors in servers increase in order to be effective. The data in the previous sections focused exclusively on energy efficiency improvements. But many datacenter workloads, however, are sensitive to latency. In order to use server level low power modes, we need to perform a careful scalability study to understand how increasing core counts impact latency of using various server level low power techniques.



(a) Apache (b) DNS (c) Mail (d) Shell

Figure 5: KnightShift provides similar energy savings to idleness scheduling algorithms but with less latency slack required.

3. Server-level Low Power Mode Scalability

We now evaluate the effectiveness of server-level low power modes with increasing core count. In this section, we will explore the results of both server-level *active* low power mode (KnightShift [26]) and idleness scheduling algorithms used in server-level *inactive* low power modes (DreamWeaver [17] and Batch [8]) on a high core count (32-core) system. We show that as core count increases, server-level active low power mode can match the energy savings of idleness scheduling algorithms, but with significantly less latency slack required. Furthermore, we show that active low power modes can offer superior energy savings at any given latency slack compared to inactive low power modes.

Energy-Latency Tradeoffs The scalability of server-level low power modes will be analyzed using energy-latency tradeoff curves. These curves show the available energy savings for a certain allowable latency slack. The *latency slack* is defined as the slack (or latency increase) allowed on 99th percentile response time. The goal of this section is to explore what latency slack is required in order for server-level low power modes to be effective. All low-power modes will incur latency impact to some extent. For workloads with stringent latency constraints (zero latency slack, for instance), the best design may be to not use any power management. Thus, this scalability study will focus on workloads that allow some level of latency slack.

The idleness scheduling algorithms in Dreamweaver and Batch can tradeoff energy and latency by adjusting the level of request queueing. When requests are queued for a longer time, there are more opportunities to place the server into a low-power idle mode, which allows for longer server sleep times. But more queueing implies longer latency. KnightShift can adjust energy-latency tradeoffs by adjusting the threshold of the switching policy. In short, KnightShift couples a low power Knight system with a high performance primary server. The Knight handles most of the low utilization requests while the primary server handles computationally demanding requests. For KnightShift to allow increased latency for higher energy savings, we can increase the threshold to switch *out of* the Knight and into the primary server. This keeps the Knight active longer at the expense of increased latency. Similarly, to decrease latency at the cost of energy savings, we can increase the threshold to switch *into* the Knight, so the primary server can stay on longer even if the utilization falls below Knight capacity.

Utilization	CPU	Memory	Disk	Other
Max	40%	35%	10%	15%
Idle	15%	25%	9%	10%

Table 1: BigHouse server power model based on [24] and [12]. Power is presented as percentage of peak power.

3.1. Methodology

To evaluate scalability, we use the BigHouse simulator [18], a simulation infrastructure for data center systems. BigHouse is based on stochastic queueing simulation [18], a validated methodology for simulating the power-performance behavior of data center workloads. BigHouse uses synthetic arrival/service traces that are generated through empirical inter-arrival and service distributions.These synthetic arrival/service traces are fed into a discrete-event simulation of a G/G/k queueing system that models active and idle low-power modes through state-dependent service rates. Output measurements, such as 99th percentile latency, and energy savings, are obtained by sampling the output of the simulation until each measurement reaches a normalized half-width 95% confidence interval of 5%. The baseline server power model used in BigHouse is shown in table 1. This model is based on component power breakdowns from HP [24] and Google [12].

We implemented the KnightShift server in BigHouse. Because BigHouse cannot accurately capture transition penalty due to statistical sampling, we assume an ideal KnightShift configuration where there are no transition delays. We will explore in detail in the next section how different switching policies with realistic transition penalties will affect overall energy and performance impact when running real-world datacenter utilization traces.

We evaluate four workload distributions, `DNS`, `Mail`, `Apache`, and `Shell`, provided with the BigHouse sim-

ulator. Each workload's load is scaled so that the modeled server within BigHouse operates at 30% average utilization, similar to average utilization in [5].

3.2. Case study with 32-core server

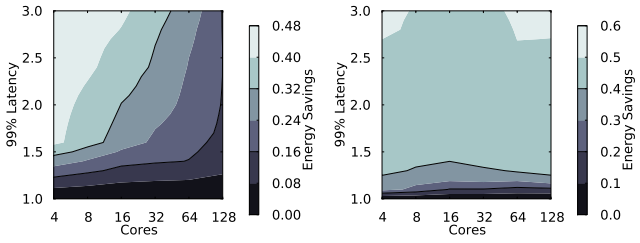
We compare energy-latency tradeoffs of the following server-level low power approaches: (1) a 30% capable KnightShift, (2) Batching [8] and (3) DreamWeaver [17].As shown in [26], the power consumption of the Knight is equal to $PrimaryServerPower * KnightCapability^{1.7}$. The quadratic assumption is based on historical data [4] which showed that power consumption increased in proportions to $performance^{1.7}$. Thus a 30% capable Knight is expected to spend 13% of the power of the primary server.

Figure 5 shows the latency vs energy savings curves of the four workloads. The latency slack shown is normalized to the workloads 99th percentile latency. The y-axis shows the energy savings possible if we are allowed to relax the 99th percentile response time by the given x-axis value.

Batching provides a nearly linear tradeoff between latency and energy, but is consistently outperformed by DreamWeaver, confirming previous results in [17]. Compared to DreamWeaver, KnightShift improves energy savings at any given latency slack. For `Mail`, KnightShift provides similar energy savings with less than half the latency slack required of DreamWeaver. For `DNS`, KnightShift provides similar energy savings with 25% less slack required. For `Apache` and `Shell`, DreamWeaver with 3x latency slack has less energy savings than KnightShift at 1.3x latency slack. *In all cases, we conclude that server-level active low power modes outperform server-level inactive low power modes at every latency slack.*

For workloads that can tolerate very large latency slack (3x or more), it may be possible to also consider the use of wimpy cluster [3, 13], which can allow power savings greater than any of the approaches compared here. But when very large latency slack cannot be tolerated, then KnightShift offers almost all of the power savings up front, with a tighter latency slack.

Power savings achievable from KnightShift saturates rather quickly with even a small latency slack. KnightShift can take advantage of all the opportunity periods for low power mode at a low latency slack. For idleness scheduling algorithms, the opportunity periods increase as the latency slack increases. The maximum savings of KnightShift saturates at ∼1.75x latency slack in most cases. This contrasts to idleness scheduling algorithms, which ramp up energy savings slowly as latency slack increases. But they never reach the maximum energy savings achievable with KnightShift. For workloads which requires latency slack even tighter than what KnightShift can provide, system-level low-power modes, both active and inactive modes, may not be the best solution and energy saving techniques may even be disregarded all together.



(a) DreamWeaver (b) KnightShift

Figure 6: Energy savings vs core count and latency slack

These results show that there must be at least a 1.5x latency slack available in order to allow server-level low power modes the opportunity to achieve a majority of their power savings potential. Essentially, *in order for server-level active low power modes to be effective, the best-case latency slack required would be 1.5x* that of the baseline. Currently under realistic conditions, the KnightShift server in [26] requires at least a 2x latency slack on average. In the next section we will explore various mode switching policies in order to try to meet the best-case latency slack time.

3.3. Sensitivity to Core Count

For this experiment we vary the number of cores in the primary server from 4 to 128 in order to explore how increasing core count affects the effectiveness of server-level low power modes.

Figure 6 shows the energy savings that can be realized across different core counts and latency slack allowed for the `Apache` workload. Results for the other workloads also follow similar trends. Figure 6a shows the possible energy savings using DreamWeaver. At low core counts, DreamWeaver can achieve significant power savings (over 40%) with relatively low latency slack (∼1.6x). But as core count increases, the potential energy savings quickly decreases and the latency slack required for similar energy savings at lower core counts increases drastically (Over 3x latency slack at 32-cores to save 40% energy!). The reason that idleness scheduling algorithms becomes less effective with core count is that they primarily rely on idle periods to exploit power savings.

Figure 6b shows the energy savings with KnightShift. Similar to inactive low power modes, significant energy savings can be achieved at low core counts. But unlike inactive low power modes, *active low power modes can sustain significant energy savings, independent of core count, and maintain a constant low latency slack.* KnightShift is not dependent on idle periods, but on low utilization periods, which remain present even at high core counts. Therefore, as long as low utilization periods exists, KnightShift can scale to any number of cores.

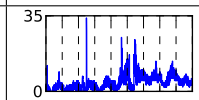
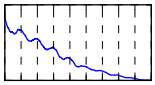
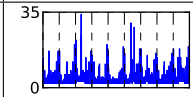
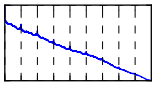
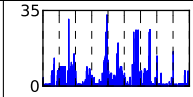
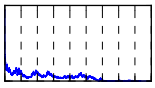
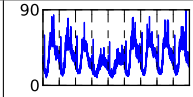
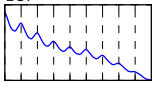
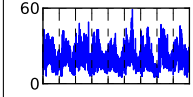
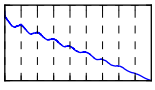
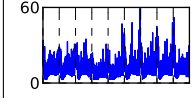
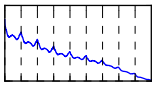
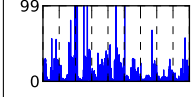
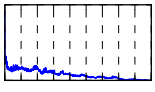
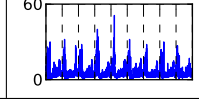
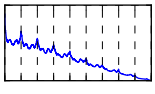
Name	Trace	Autocorrelation
aludra		
email		
girtab		
msg-mmp		
msg-mx		
msg-store		
nunki		
scf		

Table 2: Workload [26] thumbnail and autocorrelation

4. Minimizing realistic latency slack of server-level active low power mode

We have shown in the last section that server-level active low power modes have consistent opportunity, independent of server core counts, with a low latency slack requirement. But the evaluations in section 3 assumed zero transition penalty between the Knight and primary server. Under more realistic conditions that was evaluated in [26], they showed that non-bursty, low utilization workloads suffered 1.5x latency increase or less, but bursty workloads, such as `nunki`, can suffer over 6x latency penalty.

It is important for server-level low power modes to meet a low latency slack under realistic conditions in order to become a feasible alternative to cluster-level packing techniques. This would allow clusters to forego cluster-level packing techniques and enable continued cluster-wide energy proportionality improvements as shown in section 2. In this section, we will investigate in detail the cause of poor performance for certain workload categories with KnightShift. We then propose and evaluate various switching policies and its effect on energy and tail latency of KnightShift.

4.1. Workloads

We use the workload traces from [26] in our evaluation. The traces are collected over a 9 day period in an institutional datacenter from several clusters running various workloads. We show a thumbnail of the utilization traces in table 2.

Performance penalties to KnightShift can manifest in several ways. If the current utilization is low and we encounter a high utilization spike, then there would be a performance penalty when we have to wait to switch to the primary server to keep up with the high utilization requests. This is the case for very random bursty workloads (`nunki`). For workloads with periodic utilization spikes (`scf`), it may be possible to predict these periodic events to anticipate high utilization periods, but such prediction approaches were not evaluated in [26].

Another case for performance loss occurs when workloads have a very high level of variation, especially if the variation is around the Knight's capability levels. During this scenario, the KnightShift switching policy may be tricked into entering the Knight mode, when in actuality, the workload is still in a high utilization phase. This causes KnightShift to thrash between modes, which causes performance penalties during mode transitions. This is the case for workloads such as `msg-mx` and `msg-store`.

Table 2 also contains the autocorrelation of the workloads to show how strongly these utilizations exhibit daily patterns. Workloads with low predictability, such as `nunki` and `girtab`, would have low autocorrelation as shown in its autocorrelation plot. Workloads with strong daily utilization patterns, such as `msg-mmp`, `msg-mx`, `msg-store`, and `aludra` would exhibit local maxima at each day marker in their autocorrelation plots. Previous work analyzing datacenter workloads has also found strong daily correlations [11].

4.2. KnightShift Switching Policies

We will now introduce the KnightShift mode switching policies studied in this section to reduce the mode switching overhead. In order to facilitate the understanding of the strength and weaknesses of these policies, we illustrate the effect of these policies on a 15% capable KnightShift server in figure 7. For the given illustrative workload utilization pattern (7a), we show the periods where energy savings occurs (green bars on bottom of plots) and where performance penalties occur (red shaded regions). The workload utilization scenario represents a scenario with a short low utilization period (which can trick the switching policies and lead to response time increase), followed by a higher utilization and then eventually a longer low utilization period.

Aggressive Policy: The aggressive switching policy is the baseline policy used in [26]. Whenever the utilization falls below the Knight capability level, it would immediately switch into Knight mode. The server will not switch

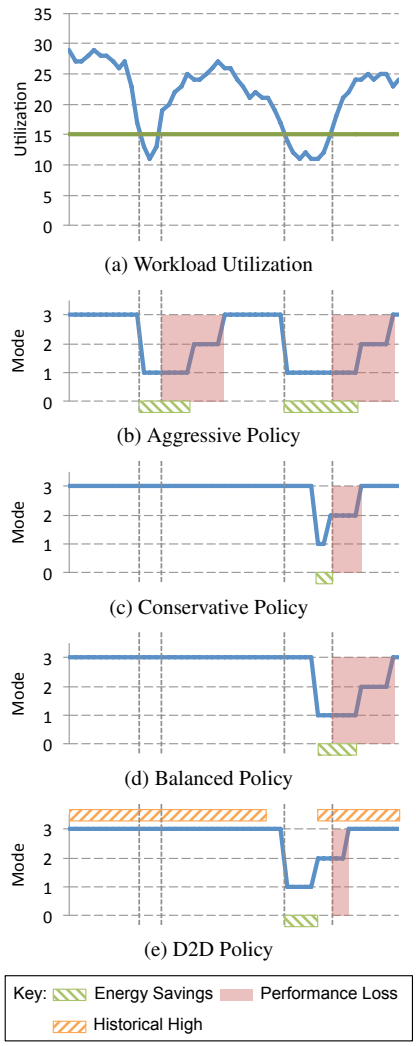


Figure 7: Performance and energy impacts of various KnightShift mode switching policies. Mode corresponds to (1) Knight mode, (2) Wakeup, and (3) Primary mode.

back to the primary server until the server experiences utilization levels beyond what the Knight can handle for a period of time equivalent to the transition time between the Knight to Primary server. As can be seen in figure 7b, the aggressive policy can save significant amount of power, but at the cost of high performance penalties. For very short low utilization periods, this simple policy can be tricked to switch into the Knight, and immediately experience utilization greater than it can handle.

Conservative Policy: This policy aims to minimize performance loss by sacrificing energy saving opportunities during short low utilization periods. A conservative policy in KnightShift would switch into the Knight only when the server's utilization level has been below the Knight's capability level for a certain amount of time. We assume this threshold to be equivalent to the wakeup transition time in all policies in this section. A transition from Knight to primary server will occur immediately upon high utilization. Figure 7c shows that the policy does not switch during short low utilization periods. Since the policy only switches when there is a long enough low utilization period, the conservative policy will avoid the performance penalty due to short low utilization periods. The performance penalty to the conservative policy would therefore be limited to when the server transitions from Knight to the Primary server. The conservative policy trade off power savings for performance by missing some opportunity to safely be in a low power state during the start of long low utilization periods.

Balanced Policy: The balanced policy aims to seek a balance between the aggressive and conservative policy by achieving energy savings with low performance impact. The balanced policy conservatively switches into the Knight and conservatively switches out of the Knight. Figure 7d shows the effect of using a balanced policy. By switching into the Knight conservatively, we avoid short low utilization periods, avoiding performance penalties due to untimely and aggressively switching into the Knight during high utilization periods. By conservatively switching out of the Knight, we are able to extract as much energy savings as possible while in Knight mode by trying to stay in an energy saving state as long as possible. This also makes this policy stay in a low power state when faced with a utilization spike, saving energy at the cost of performance.

Day to Day (D2D) Policy: This simple heuristic policy aims to use the insight derived from the autocorrelation plots of each workload shown in table 2. This policy demonstrates the possible effect of a more sophisticated switching policy compared to the other policies in this section. By looking at the binary historical utilization levels (either high or low), we can anticipate high utilization periods. For this policy, we use the aggressive policy as the base policy to build off of and extend it with knowledge of past historical high periods. We can only switch into the Knight aggressively only if there is not a historical high from the previous day. Similarly, if currently in the Knight mode, a historical high is detected, we will preemptively switch into the primary server state to anticipate a high utilization period regardless of current utilization.

In our study, we log minute-level granularity of historical high periods. Because it's unlikely for high/low utilization to occur daily at exactly the same minute, we expand the window of historical high periods by +/-15 minutes. We empirically selected 15 minutes as it provides a good balance between energy savings and performance impact. This allows nearby high utilization periods to merge into a larger high utilization window. This prevents KnightShift from switching aggressively into the Knight during this period, which normally led to performance penalty as seen in figure 7b. Figure 7e shows the past historical highs marked

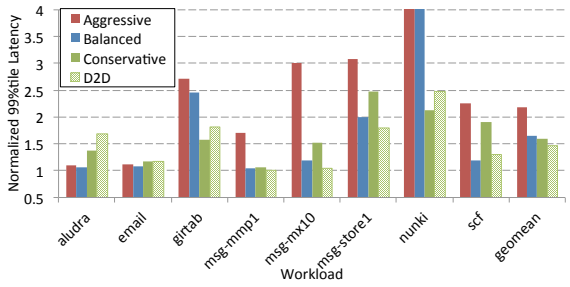
with an orange bar on top of the plot. By detecting historical high periods, we avoid switching during short low utilization periods and anticipate future high utilization periods, avoiding significant performance penalty, while still achieving reasonable energy savings.

4.3. Switching Policy Evaluation

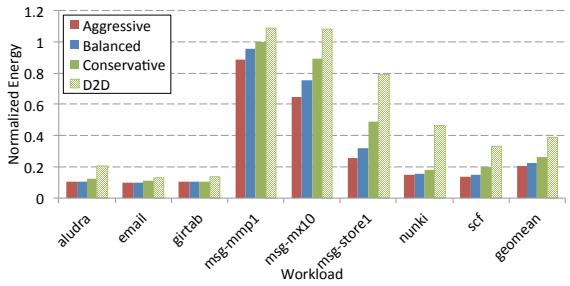
In this section we use the evaluation methodology presented in section 2 with 1 server. The results in this section is compared to a baseline machine running without KnightShift. We found that the performance of the baseline server is also representative of the tail latency observed in cluster-level techniques shown in section 2. In cluster-level packing techniques, there are spare servers providing extra capacity to absorb utilization spikes, leading to minimal performance impact of cluster-level packing techniques. In order for server-level low power modes to be competitive with cluster-level packing techniques, it is necessary that server-level techniques meet the best-case latency slack, as shown in section 3, to minimize tail latency impact.

Effect on Latency: Figure 8a shows the 99th percentile latency normalized to the 99th percentile latency of the baseline server. Although the aggressive policy has the highest geometric mean latency, there are several workloads that actually performs the best using this policy. In particular, workloads that tend to stay low with rare utilization spikes (`aludra`, `email`) seems to benefit the most from this policy. The conservative policy benefits workloads (`nunki`, `girtab`) that tend to be random (low autocorrelation). Workloads with high utilization variations (`msg-mx`, `msg-store`, `scf`) benefits most from a balanced policy. Both `msg-mmp` and `msg-mx` have latency of near 1 because the policies realize that these workloads tend to always be in a high utilization phase, and therefore does not go into knight mode. For workloads with daily patterns observable from their autocorrelation plot (`msg-mmp`, `msg-mx`, `msg-store`, `scf`) all benefits the most from the Day-to-Day switching policy. Using a very simple 1-day history heuristics, KnightShift is able to lower the overall geometric mean latency to about 1.5x the baseline latency. This is in line with the best-case energy-latency tradeoff curves in section 3. *Therefore, it is possible to meet the best-case latency slack, even under realistic conditions.*

Effect on Energy: Figure 8b shows the normalized energy consumption normalized to the baseline server. Note that the energy consumption values presented here for KnightShift cannot be quantitatively compared to the values presented in section 3 due to entirely different simulation methodologies and workloads. For all scenarios, the aggressive policy saves the most energy (79.5% geometric mean). As expected, the balanced policy's energy usage (77.8%) falls in between that of the aggressive policy and the conservative policy (73.7%). Aggressive saves the most power, but at cost of highest latency impact. The D2D policy mean-



(a) Switching policy effect on latency



(b) Switching policy effect on energy

Figure 8: Effect of switching policy on latency and energy consumption. The best-case latency slack (section 3) is achievable under realistic conditions.

while saves the least amount of power due to cautiously staying in the primary server mode to anticipate historical high periods. In certain workloads, such as `msg-mmp` and `msg-mx`, KnightShift actually consumes more energy than the baseline server as these workloads are high utilization workloads and does not offer opportunity for KnightShift's low power mode. The extra energy used is due to the overhead of the Knight remaining on all the time. *Overall, the conservative and balanced policy provides the best balance of latency and energy savings.*

Summary: In this section, we showed that server-level active low power modes, with corresponding policies, can achieve the best-case latency slack needed to achieve energy savings under realistic conditions. Server-level techniques can be competitive with cluster-level packing techniques, requiring only a small latency slack. In previous section, we showed that while server-level inactive low power modes became ineffective due to core count, server-level active low power modes can remain effective even with increasing server core count. By overcoming these two challenges, server-level low power modes can be an attractive alternative over cluster-level packing techniques as future servers continue to improve their energy proportionality.

5. Conclusion

While cluster-level packing techniques provided an answer to cluster-wide energy proportionality in the past, the continuing improvements to individual server energy

proportionality is threatening to disrupt this convention. Cluster-level packing techniques can now actually limit cluster-wide energy proportionality. As we near a turning point, it may be favorable to forgo cluster-level packing techniques and rely solely on server-level low power modes. In order for server-level low power modes to become practical, there must be improvements to the latency slack required for server-level low power modes to be effective. We have shown that server-level *active* low power modes can provide consistent energy savings at a low latency slack *independent* of server core counts, unlike server-level *inactive* low power modes. Furthermore, we have shown that with the right mode switching policies, server-level active low power modes, such as KnightShift, can meet the best-case latency slack under realistic conditions. By solving these issues, we demonstrate the potential for server-level low power mode use in practice.

Acknowledgment

We would like to thank our shepherd, Tom Wenisch, and the anonymous reviewers for their valuable comments. This work was supported by DARPA-PERFECT-HR0011-12-2-0020 and NSF grants NSF-1219186, NSF-CAREER-0954211, NSF-0834798.

References

[1] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *NSDI'09*, 2009.

[2] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini, T. Yang, D. Franklin, and F. T. Chong. Barely alive memory servers: Keeping data active in a low-power state. *J. Emerg. Technol. Comput. Syst.*

[3] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *SOSP '09*.

[4] M. Annavaram, E. Grochowski, and J. Shen. Mitigating amdahl's law through epi throttling. *SIGARCH Comput. Archit. News*, 33(2), May 2005.

[5] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12), dec 2007.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005.

[7] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: active low-power modes for main memory. In *ASPLOS '11*, 2011.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, 2003.

[9] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07*, 2007.

[10] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4), Nov. 2012.

[11] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Workload Characterization 2007 IISWC 2007 IEEE 10th International Symposium on*, 2007.

[12] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[13] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. *SIGARCH Comput. Archit. News*, 38(3), June 2010.

[14] K. T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. C. Lee, and M. Horowitz. Rethinking DRAM Powermodes for Energy Proportionality . In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45 '12, 2012.

[15] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS '09*. ACM, 2009.

[16] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *ISCA '11*. ACM, 2011.

[17] D. Meisner and T. F. Wenisch. Dreamweaver: architectural support for deep sleep. In *ASPLOS '12*. ACM, 2012.

[18] D. Meisner, J. Wu, and T. F. Wenisch. Bighouse: A simulation infrastructure for data center systems. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, 2012.

[19] R. Nathuji and K. Schwan. VirtualPower: coordinated power management in virtualized enterprise systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007.

[20] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: Coordinated multi-level power management for the data center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, 2008.

[21] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. *SIGARCH Comput. Archit. News*, 34(2), May 2006.

[22] C. Subramanian, A. Vasan, and A. Sivasubramaniam. Reducing data center power with server consolidation: Approximation and evaluation. In *High Performance Computing (HiPC), 2010 International Conference on*, 2010.

[23] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble. In *HotPower'08*, 2008.

[24] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD '10*, 2010.

[25] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 2005.

[26] D. Wong and M. Annavaram. Knightshift: Scaling the energy proportionality wall through server-level heterogeneity. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45 '12, 2012.

[27] www.spec.org/power_ssj2008/. Spec power_ssj2008, 2012.