KnightShift: Scaling the Energy Proportionality Wall Through Server-Level Heterogeneity

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

Abstract

Server energy proportionality has been improving over the past several years. Many components in a system, such as CPU, memory and disk, have been achieving good energy proportionality behavior. Using a wide range of server power data from the published SPECpower data we show that the overall system energy proportionality has reached 80%. We present two novel metrics, linear deviation and proportionality gap, that provide insights into accurately quantifying energy proportionality. Using these metrics we show that energy proportionality improvements are not uniform across various server utilization levels. In particular, the energy proportionality of even a highly proportional server suffers significantly at non-zero but low utilizations. We propose to tackle the lack of energy proportionality at low utilization using server-level heterogeneity. We present KnightShift, a server-level heterogeneous server architecture that introduces an active low power mode, through the addition of a tightly-coupled compute node called the Knight, enabling two energy-efficient operating regions. We evaluated KnightShift against a variety of real-world datacenter workloads using a combination of prototyping and simulation, showing up to 75% energy savings with tail latency bounded by the latency of the Knight and up to 14% improvement to Performance per TCO dollar spent.

1. Introduction

Energy consumption of datacenter servers are a critical concern. Server operating energy costs comprise a significant fraction of the total operating cost of datacenters. However, many servers operate at low utilization and still consume significant energy due to the lack of ideal energy proportionality [6].

Server consolidation [7, 8] can boost utilization on some servers while allowing idle servers to be turned off, improving energy proportionality at the datacenter level. Unfortunately, server shutdown is not always possible due to data availability concerns and workload migration overheads. When server shutdown is impractical, as is the case in many industrial data centers [3, 26], system-level energy proportionality approaches must be explored.

Energy proportionality improvements of various server components [10, 30], such as CPUs and memory, has fueled the improvements of overall system efficiency. Energy proportionality of current systems has reached around 80%. While 80% seems reasonable, the primary concern today is that energy proportionality improvements have not been uniform across different utilizations. The problem of disproportionality is particularly acute at non-zero but low server utilization. Since no single component dominates server energy usage [31], holistic system-level approaches must be developed to improve energy proportionality particularly at low utilization regions.

Several system-level power saving approaches have focused on reducing the power consumption during idle periods [24]. Researchers then focused on increasing the length of idle periods by queueing requests [26] or by shifting I/O burden directly to disk and memory [2, 3]. However, as multicore servers becomes dominant, idle periods are virtually nonexistent [26, 34]. Even as idle periods become rare, servers still spend a significant fraction of their execution time operating at low utilization levels. Thus there is a critical need to develop active low-power modes that exploits low-utilization periods to continue improving server-level energy proportionality across the entire utilization range.

This paper addresses this critical need by proposing KnightShift, a server-level energy proportionality technique. This work makes the following contributions:

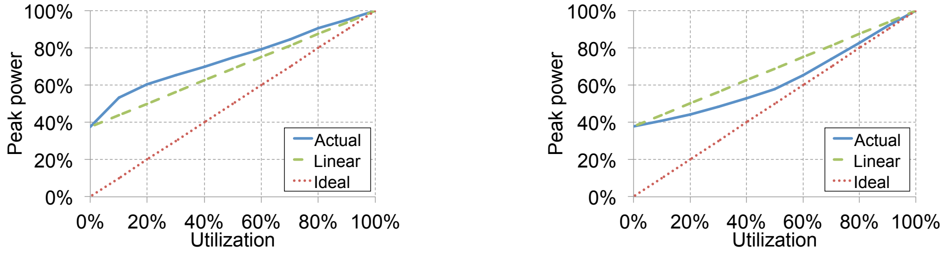
Metrics to Identify Disproportionality(§2): We propose metrics to evaluate energy proportionality and to identify sources of disproportionality. Using data from historical SPECpower [35] results of 291 servers, we show that commonly used metrics such as dynamic range are inappropriate due to the lack of linearity in energy consumption across different utilizations. We present a metric for measuring linearity of energy consumption across different utilizations. Using the linearity metric we show that the proportionality gap is much wider at lower utilization than at idle or higher utilization.

Energy Proportionality Trend Analysis(§3): From historical SPECpower data we show the existence of an energy proportionality wall due to the lack of improvements to the dynamic range and poor energy efficiency at low server utilization periods. Previous work (§4) only targeted improvements to the dynamic range by improving idle power. In order to continue improving energy proportionality, we must improve the linearity of the server's energy proportionality curve, especially at lower utilization where the majority of the proportionality gap exists.

KnightShift(§5): We present KnightShift, a server-level heterogeneous server architecture that introduces an active low power mode to exploit low-utilization periods. By fronting a high-power primary server with a low-power compute node, called the Knight, we enable two energy-efficient operating regions. We show that KnightShift effectively improves energy proportionality and linear deviation of servers in §6. We present evaluation results of KnightShift in §7 and explore TCO impact in §8.

2. Measuring Energy Proportionality

In order to understand energy proportionality trends, we must first quantify energy proportionality. Figure 1 illustrate the power usage of two servers over their operating utilization, called the *energy proportionality curve*. The dotted line shows the *ideal* energy proportionality curve of a server. The dashed line shows the *linear* energy proportionality curve by interpolating idle and peak power. The solid line shows the *actual* server energy proportionality curve. The data presented in this figure are obtained from measurements on real servers reported to SPECpower (more detailed analysis of SPECpower data is provided later).



(a) Superlinear EP

(b) Sublinear EP

Figure 1: Energy Proportionality (EP) curve. The dotted, dashed, and solid lines shows the ideal, linear, and actual server EP curve, respectively.

Dynamic Range: The *dynamic range* (DR) metric is commonly used as a first order approximation for energy proportionality. The dynamic range of a server is given by:

$$DR = \frac{Power_{peak} - Power_{idle}}{Power_{peak}} \tag{1}$$

where $Power_{peak}$ is the peak power at 100% utilization and $Power_{idle}$ is the idle power at 0% utilization. In Figures 1a and 1b, the DR for both servers are the same at 60% . An ideal energy proportional system would have DR of 100%. DR only accounts for peak and idle power usage and does not account for power usage variations across different utilizations. Since most servers are rarely fully utilized or fully idle, DR is a poor measurement of the server's actual proportionality. For example, assume that both servers in Figure 1 consume 100W at peak power. If each server experiences utilization distribution similar to Google servers reported in [6], then Server A (on the left) would consume on average 28% more power (68.6W vs 52.6W) compared to Server B (on the right), even though they both have the same dynamic range.

Energy Proportionality: To accurately quantify energy proportionality, we must account for intermediate utilization power usage. The *energy proportionality* (EP) of a server (proposed in [29] and adapted for this paper) is given by:

$$EP = 1 - \frac{Area_{actual} - Area_{ideal}}{Area_{ideal}} \tag{2}$$

where $Area_{actual}$ and $Area_{ideal}$ is the area under the server's actual and ideal energy proportionality curve, respectively. Note that if $Area_{actual} = Area_{linear}$, then EP would equal DR. Therefore, DR is a good measurement of energy proportionality only if a server is *linearly energy proportional*, however, this is not the case in most servers. For example, the EP of Server A and B is 53% and 74%, respectively. Although the DR of both servers is 60%, their EP values *differ by over 20%*. Compared to DR, EP provides a more accurate metric in determining how energy efficient a server is. Energy proportionality is a function of the dynamic range and the linearity of the energy proportionality curve. Thus to accurately account for energy proportionality, one has to account for the amount of deviation from linearity within the server's energy proportionality curve.

Linear Deviation: We define *Linear Deviation* (LD) as a measure of the energy proportionality curve's linearity. Linear Deviation is given by:

$$LD = \frac{Area_{actual}}{Area_{linear}} - 1 \tag{3}$$

A server is considered *linearly energy proportional* if LD = 0, *superlinearly energy proportional* if LD $>$ 0, and *sublinearly energy proportional* if LD $<$ 0. Figure 1a and 1b shows a proportionality curve with superlinear and sublinear energy proportional system, respectively. Superlinear energy proportional servers have EP $<$ DR , while sublinear energy proportional servers have EP $>$ DR. This can be proven by equation 2 where $Area_{+LD} > Area_{linear} > Area_{-LD}$, therefore $EP_{+LD} < EP_{linear} < EP_{-LD}$, where $EP_{linear} = DR$.

Proportionality Gap: The *Proportionality Gap* (PG) is a measure of deviation between the server's actual energy proportionality and the ideal energy proportionality at individual utilization levels. PG allows us to quantify the disproportionality of servers at a finer granularity compared to EP to better pinpoint the causes of disproportionality. PG at utilization level $x\%$ is given by:

$$PG_{x\%} = \frac{Power_{actual@x\%} - Power_{ideal@x\%}}{Power_{peak}} \tag{4}$$

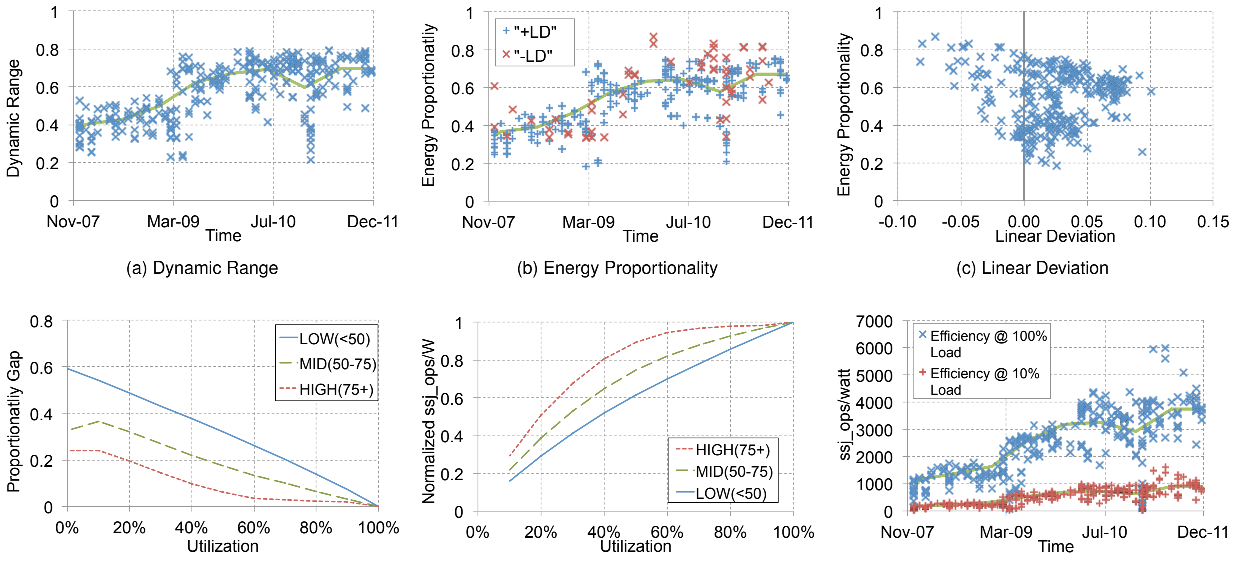
For an ideal energy proportional server, the PG for all utilization levels is 0. For superlinearly proportional systems, like Server A, PG is very large at zero utilization and it continues to grow for some time before it starts to shrink. For sublinearly proportional systems, like Server B, PG is very large at zero utilization but it continues to decrease with utilization.

3. Energy Proportionality Trends

To understand trends in energy proportionality, we analyze the submitted results of SPECpower [35] for 291 servers from November 2007 to December 2011. These servers are a representative mix of server configurations in current use. They feature servers with various vendors, form factors, and processors. The SPECpower benchmark evaluates the power and performance characteristics of servers by measuring the performance and power consumption of servers at each 10% utilization interval. These trends are shown in Figure 2 and are discussed below.

Dynamic Range: Figure 2a plots the dynamic range of servers along with the median trend line. Each data point corresponds to one server whose SPECpower results were posted on a given date. Overall, DR improved from about 50% to 80% from 2007 to 2009. From 2009 onward, DR stagnated at 80%. Although the best DR is 80%, half of new servers today still have DR less than 70%. Even in 2011, there are still new servers with DR less than 40%. We can surmise that achieving 100% dynamic range is very difficult due to energy disproportional and energy inefficient components such as power supplies, voltage converters, fans, chipsets, network components, memory, and disks.

Energy Proportionality: Figure 2b shows that EP trends are similar, but not identical, to DR trends. Clearly, EP has also stalled at around 80%. This *energy proportionality wall* is mainly due to the lack of DR improvement. Each server's EP data point is classified



(a) Dynamic Range

(b) Energy Proportionality

(c) Linear Deviation

(d) Proportionality Gap

(e) Energy Efficiency

(f) Efficiency Growth

Figure 2: Energy Proportionality Trends

as either superlinear (+LD) or sublinear (-LD) proportional based on their SPECpower data. It is important to draw attention to a few data points where EP > 80%, although no servers have a DR above 80%. Recall that the only way a server can have EP > DR is for that server to have sublinear energy proportionality (-LD). A sublinearly energy proportional server consumes less power than a linearly proportional server. Hence, it can have higher EP than DR. Thus, those few servers with EP > 80% in Figure 2b have sublinear energy proportionality. Note that -LD does not always imply high EP. In particular, energy proportionality is affected by two components: dynamic range and the linear deviation. If DR can be improved, then LD improvements have a secondary impact on overall EP. But as DR improvements hit a wall, the only way to improve EP moving forward is to improve LD.

Linear Deviation: Figure 2c shows the relationship between linear deviation and energy proportionality. Unfortunately, the data shows that the majority of servers (at least 80%) are superlinearly proportional (+LD). Hence, there lies potential to improve energy proportionality in current servers by improving their linear deviation.

Proportionality Gap: Figure 2d shows the average proportionality gap of servers at various utilization levels. The curves, from top to bottom, represent servers with low EP (<50%), medium EP (50-75%), and high EP (>75%). Irrespective of the EP level the striking feature is that all servers suffer large proportionality gap at low utilizations. Furthermore, as EP increases, it becomes clear that the majority of the proportionality gap occurs at lower utilizations. As EP improves, *energy disproportionality at lower utilization will be the main obstacle to achieving perfectly energy proportional systems*. Unfortunately, due to limited information reported in SPECpower results, we cannot gain a clear insight as to the fundamental causes of proportionality gap at lower utilization and thus this question remains an open research problem.

Energy Efficiency: We defined energy efficiency as ssj_ops/Watt from the SPECpower data. Figure 2e shows that energy efficiency is strongly correlated with the proportionality gap. The curves, from top to bottom, represents servers with high, medium, and low EP, respectively. Due to the large proportionality gap at low utilization, server energy efficiency is about 30% of the peak efficiency even for servers with relatively high EP. Hence, even if the overall EP of a server improves over time, the energy efficiency will still suffer at low utilizations unless the proportionality gap at low utilizations is reduced. Otherwise, even the highest EP servers can run efficiently only at high utilizations. In order to improve energy efficiency, we should improve efficiency at lower utilization. Unfortunately, as Figure 2f shows, improvements to efficiency at higher utilization has outpaced improvements at lower utilization.

Overcoming the EP wall: In order to improve the energy efficiency of servers, we cannot solely rely on improvements to dynamic range, as has been the case in the past. Therefore, we cannot concentrate on energy efficiency improvements at peak and idle only. As dynamic range is now static, we must focus on improving the linear deviation. As shown previously, servers operate in two distinct energy proportional regions. Servers tend to be perfectly proportional at high utilization (>50%), while disproportional at low utilization (<50%). Therefore, in order to gain the most benefits, we must focus our efforts in the low utilization region. Furthermore, processors are no longer the major consumer of power in servers [31]. In order to reduce energy consumption, new server-level solutions that tackle proportionality gap at low utilizations are needed.

4. Related Work

Power and energy related issues in the context of large scale datacenters have become a growing concern for both commercial industries and the government. Barroso [6] showed that energy-proportionality is a chief concern since most enterprise servers operate at low average utilizations. These concerns have become the source of much active research in the energy proportional computing space. Numerous studies have examined energy efficiency approaches to servers in datacenter settings. These approaches can be classified along three dimensions: spatial granularity (Granularity), the period in which the low power mode is active (Period), and the ability for the low power mode to perform work (Active/Inactive). The granularity refers to whether the low power mode work at the cluster, server, or component level. The period in which the low power mode is active refers to the region of operation that the low power mode exploits. Low power modes can exploit either idle periods (0% utilization), or low

Granularity	Cluster-level		Server-level		Component-level	
Period	**Idle**	**Low Utilization**	**Idle**	**Low Utilization**	**Idle**	**Low Utilization**
Active Low power		Consolidation & Dynamic Cluster Resizing [7, 8]	Somniloquy [1] Barely-alive Servers [3]	KnightShift		DVFS MemScale [10] Heter. Cores [21, 14, 13, 5]
Inactive Low power			Shutdown PowerNap [24]		DRAM Self-refresh Core Parking Disk Spin down	

Table 1: Classification of Server Low-power modes

utilization periods. The ability to perform work refers to whether the low power mode allows the system to continue processing requests. For inactive low power modes, the system cannot processes requests. For active low power modes, the system can still process requests, possibly with lower capability and performance. For example, if a low power mode is an active low power mode and exploits low utilization periods, it means that the low power mode is activated during low utilization periods and can still perform work. Using the three dimensional classification Table 1 bins the most relevant prior work which we will briefly describe next .

Cluster-level techniques: Common techniques such as consolidation and dynamic cluster resizing [7, 8] concentrate workload to a group of servers to increase average server utilization and power off idle machines, improving efficiency and lowering total power usage. Although beneficial, these techniques are not suitable for many emerging workloads in today's datacenter settings. For direct-attached storage architectures or workloads with large distributed data sets, servers must remain powered on to keep data accessible. Furthermore, due to the large temporal granularity of these techniques, they cannot respond rapidly to unanticipated load as it could take minutes to migrate tasks with very large working sets. Under these circumstances, server consolidation is not a viable solution. Our proposed solution will allow significant energy savings even when servers are required to actively operate at low utilization.

Component-level techniques: Component-level energy saving techniques for CPU, memory, and disk covers both active and inactive low-power modes. Active low-power techniques improves the energy-proportionality of components by providing multiple operating efficiencies at different utilization levels. Heterogeneous cores [5, 13, 14, 21], such as Tegra 3 and ARM big.LITTLE, can switch to low-power efficient cores during low-utilization periods, while DVFS and MemScale [10] scales the frequency and power of components depending on utilization levels. Furthermore, inactive low-power techniques, such as DRAM self-refresh, core parking and disk spin down can improve idle power consumption of these components. Most dynamic range improvements seen to date are driven primarily by processor energy efficiency gains. But going forward, no single component dominates overall power usage [31], which may limit the potential of component-level techniques in the future.

Server-level techniques: Server-level techniques aim to put the entire server into a low-power mode. Previous techniques aimed to improve energy efficiency by increasing the dynamic range through lowering the idle power usage and extending the time a system stays in idle. PowerNap [24] exploits millisecond idle periods by rapidly transitioning to an inactive low-power state. DreamWeaver [26] extends PowerNap to queue requests, artificially creating and extending idle periods. Barely-alive servers [3] place the server in a low-power state, but extends idle periods by keeping memory active to process remote I/O requests. Similarly, Somniloquy [1] allows idle computers to supports certain application protocols, such as download and instant messaging. As the number of processors in servers increase, idle periods will become increasingly rare [26, 34]. Thus *active low-power modes that can efficiently operate at low-utilization levels will be the only practical server-level energy saving technique in the future*. Current literature lacks work that exploit low-utilization opportunities. As the data in Section 3 showed, it is critical to tackle the lack of energy efficiency during low-utilization periods. Our work, KnightShift, fills this important gap.

Low-power design: Wimpy nodes [4] aims to save power by running low-power energy-efficiency nodes. Wimpy clusters are limited to workloads that can tolerate higher latency, but may degrade QoS during traffic spikes, requiring over-provisioning [28, 15]. Heterogeneous clusters [9] of brawny and wimpy cores also suffers the same issues as consolidation and task migration. In KnightShift, we can dynamically switch modes to handle latency demands, without the overhead of consolidation due to a tightly-coupled disk subsystem.

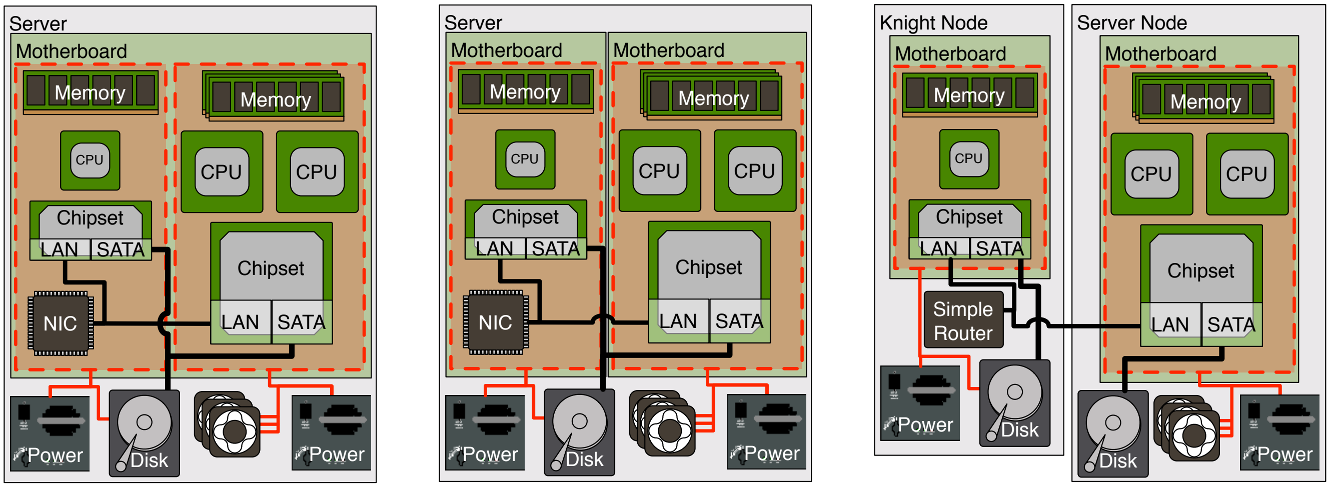
5. KnightShift

In this section we introduce KnightShift, a heterogeneous server-level architecture to reduce the proportionality gap of servers at low utilization. KnightShift fronts a high-power primary server with a low-power compute node, called the Knight, enabling two energy-efficient operating regions. We define *Knight capability* as the fraction of throughput that Knight can provide compared to the primary server. To the best of our knowledge, KnightShift is the first *server-level active low-power mode solution to exploit low-utilization periods*. The fundamental issues limiting energy proportionality have been lack of improvement to dynamic range and linear deviation. While previous techniques only targeted dynamic range, KnightShift extends previous techniques by also targeting linear deviation.

A KnightShift system consists of three components:

I **KnightShift hardware:** The KnightShift hardware consists of a low-power low-performance compute node, called the Knight, paired with a high-power high-performance server. Both the Knight and primary server can be independently powered on and off. Both the Knight and primary server share a common data disk and are able to communicate with one another through traditional network interface. In section 5.1, we will introduce three possible implementations of KnightShift.

Due to low-power demands we assume the Knight has less memory than the primary server. However, certain workloads require large memory-resident datasets, such as scale-out workloads [12], and cannot tolerate smaller memory. These workloads can still benefit from KnightShift by alternatively using low-power mobile memory [23], therefore still benefiting from overall reduced energy savings. Current server motherboards are typically not built to accommodate low-power mobile memory while a Knight can use such a memory type.



(a) Board-level

(b) Server-level

(c) Ensemble-level (Prototype)

Figure 3: Three proposed implementations of KnightShift. In a board-level implementation the primary server and Knight are integrated into the same motherboard. In a server-level implementation the Knight is a separate add-on board that attaches to SATA port, converting commodity servers into KnightShift systems. In an ensemble-level implementation, only commodity parts are used.

II **System software:** The system software enables several key functionalities required for KnightShift, such as disk sharing, network configuration and remote wakeup of compute nodes. Most operating systems already support the required system software functionality. In section 5.1 we will describe the specifics of system software required to support KnightShift.

III **KnightShift runtime:** The KnightShift runtime is the new software layer that is built specifically for the purpose of operating KnightShift. This runtime layer monitors utilization, makes mode switching decisions, redirect requests between the Knight and the primary server, and coordinates disk access rights to ensure data consistency. We will discuss this runtime in detail in section 5.2 and present our prototype implementation in section 7.1.1.

5.1. KnightShift Implementation Options

We propose three implementations of KnightShift as shown in Figure 3. The preferred choice for implementing KnightShift depends on the usage scenario and level of integration supported by system designers.

Board-level integrated KnightShift: Board-level integrated KnightShift integrates the primary server and Knight onto the same motherboard. Both Knight and primary server have independent memory, CPU, and chipsets. To allow each node to power on/off independently, the motherboard is separated into two power domains (designated by the dotted box). The Knight's power domain comprises of it's memory, CPU, chipset, ethernet, and disks. The Knight's power domain is always on but the primary server's power status is controlled by the Knight. The Knight is capable of remotely waking up the primary server. Existing technology such as wake-on-lan can be used to support remote wakeup. Using wake-on-lan, when the primary server is off, the Knight can send a "magic" packet to the primary server's network interface which in turn will wake up the primary server. All three proposed implementations use wake-on-lan for remote wakeup.

Networking is provided through sideband ethernet, allowing two devices to be exposed through a single physical port to external servers. Both the Knight and primary server would have their own IP address, but only the Knight's IP address would be publicly available. This allows the KnightShift server to appear as a single server on the network, eliminating additional network overheads to adopt KnightShift servers.

Disks are shared between the primary server and Knight through a shared SATA connection. Since SATA currently supports hot-plugging, the system designer can add switching logic to route SATA requests between the primary server and Knight.

Server-level integrated KnightShift: In a server-level integrated KnightShift configuration, the Knight resides on a separate independently powered motherboard. The only shared components between the Knight and primary server is the network and disk. We envision that this approach can be implemented by intercepting the SATA interface and building a Knight which can fit as a hard drive module within the primary server. Hard drive mounts are designed to fit various hard drive sizes. For example, 3.5inch drives comes in 19mm or 25.4mm heights. By using 19mm height drives or 2.5inch drives, we can integrate the Knight into the unused space on the 3.5inch mount. This approach is feasible even today as some potential Knight candidates are as small as credit cards [19].

Since the Knight remains on at all times, it is exposed to the outside world as the only server. Thus, the primary ethernet connection will be on the Knight board. The existing primary server's ethernet is then connected into the Knight board. Thus this approach requires one extra internal ethernet connection compared to board-level integration. This implementation allows us to convert any commodity server to a KnightShift-enabled server without using additional space.

Ensemble-level KnightShift: The ensemble-level implementation uses only commodity parts with no changes to hardware. By using a primary server and a Knight based on commodity computers (such as nettops), a KnightShift system can be implemented. This is the prototype that we will use to evaluate KnightShift in section 7.1. Disk sharing is fulfilled through NFS, with the Knight acting as the NFS server and the primary server mounting the NFS drive. This allows data to persist when the primary server is off. Since the Knight acts as the NFS server this approach requires the Knight to always be on. A router is used to network the Knight and primary server. To the outside world only the Knight's IP address is exposed. The primary

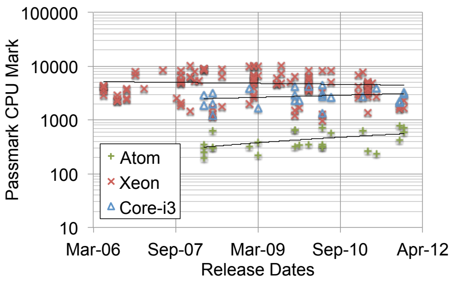


Figure 4: Performance trends of commercial systems.

server communicates to the outside world through the Knight.

Board-level implementation requires the least amount of space, but requires several modifications to the system design. The server-level implementation allows commodity servers to be KnightShift-enhanced, with minimal space requirements. The ensemble-level is the simplest to implement with commodity parts. But this solution can be expensive and may need additional rack space in a data center.

5.2. KnightShift Runtime

In the above section we presented three choices for implementing KnightShift and the basic system software needed for remote wakeup, disk and network sharing. The KnightShift functionality is implemented in a special purpose runtime layer called the KnightShift runtime. The runtime serves the following purposes: 1) Monitor server utilization, 2) Decide on when to switch between Knight and primary server using mode switching policies, 3) Ensure data consistency on shared disk data, 4) Coordinate mode switching, 5) Redirect requests to active node.

Monitoring server utilization and mode switching policies: An essential part of KnightShift is the ability to monitor the utilization of the primary server and Knight to make mode switching decisions. Server utilization monitoring can be carried out simply through the Linux kernel or through third-party libraries.

Whenever the primary server's utilization is low, the Knight will put the primary server to sleep and handle service requests. Whenever the Knight's utilization is too high, it does a remote wakeup of the primary server which then handles service requests. In this paper, our primary goal is to introduce the benefits of KnightShift and thus we use a simple switching policy to determine when to switch modes. In order to maximize power savings, we have to maximize the amount of time that we spend in the Knight. To do so, our simple switching policy aggressively switches into the Knight, and conservatively switches to the primary server. For example, if the Knight is 20% capable, the KnightShift runtime will switch to the Knight whenever the primary server utilization falls below 20%. KnightShift switches back to the primary server only when the Knight's utilization exceeds 100% for at least the amount of time it takes to switch between Knight and primary server, called the transition time.

By maximizing energy savings, we may negatively impact performance as we may stay in the Knight mode during periods where the Knight cannot handle the requests, causing increased response time. Although it is outside the scope of this work, by using a more balanced switching policy, through predicting high utilization periods or carefully chosen timeouts, KnightShift may provide a better balance between energy savings and performance [34].

Data consistency and coordinating mode switching: Recall that in all three implementation the Knight and primary server share the disk data needed for processing service requests. Hence, whenever mode switching is activated, the compute node that is shutting down must flush any buffered disk writes that are cached in memory back to disk and unmount the disk. This allows the complementary node to mount the disk and operate on consistent data. The KnightShift runtime enforces this consistency by coordinating disk access sequence between the two nodes. In section 7.1 we detail our prototype KnightShift system where coordination is carried out through a set of scripts communicating using message passing.

Redirecting requests: There are many ways to forward incoming requests to the active compute node. One approach is to run a simple load balancer software on the Knight, which would require the Knight to remain always on. We take this approach in our prototype KnightShift implementation in section 7.1. It would also be possible to use a hardware component which redirects requests.

5.3. Choice of Knights

We originally considered three options for the Knight: ARM, Atom, and Core i3-based systems. It is currently not feasible to use ARM based systems as a Knight because its capability level (<10%) is simply too low and does not provide ample opportunities to switch to Knight mode. With the emergence of server-class ARM processors, ARM may become a viable Knight option in the near future.

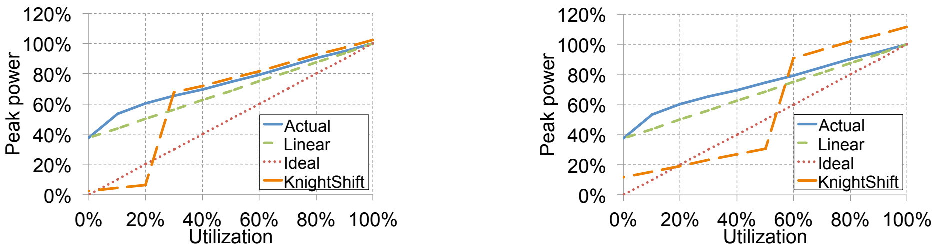
Figure 4 shows the performance growth of Atom and Core i3 as potential Knights compared to a Xeon based server as the primary server. The performance data was obtained from Passmark CPU Mark [18]. Most Atom based systems have one order of magnitude lower capability than a Xeon based server and in the best case they have 20% capability. The performance of Core i3 on the other hand, is within 50% of Xeon based server. Thus Atom and Core i3 can provide Knight capability of up to 20% and 50%, respectively. Although Core i3 based Knights use ~4x more power than Atom based Knights, Core i3 offers more opportunity for the Knight to handle requests from the primary server. In our prototype implementation we used only an Atom based Knight due to limited hardware budget.

Mixed ISA: In our current prototype, all the Knight choices run x86 ISA. Additionally, we ran a fully functional KnightShift prototype using x86+ARM and we didn't encounter any functional difficulties. Many popular applications, such as java, apache and mysql already have ARM binaries. As ARM becomes more powerful and prevalent, mixed ISA KnightShift systems may even become the norm. While the ARM based Knight ran perfectly well in terms of functionality, the latency overhead was too high. Hence we do not consider mixed ISA implementation in the rest of the paper.

6. A Case for Server-level Heterogeneity

In this section we show the potential benefits of KnightShift on top of current production systems. We selected all 291 servers from the SPECpower results and studied how various energy proportionality metrics are affected if that server was enhanced with a Knight. Recall that we define *Knight capability* as the fraction of throughput that the Knight can provide compared to the primary server. By assuming the Knight was created with the same technology as the primary server, the peak and idle power of the Knight, with capability C, can be obtained by theoretically scaling the power using the equation $Power_{Knight} = C^{1.7} * Power_{Primary}$ [5]. For example, if the primary server operates between 100W (idle power)-200W (peak power at 100% utilization), then a 50% capable Knight will operate from 31-62W. We assume the Knight is linearly proportional (LD=0) between its idle and peak power.

Figure 5 shows the effect of KnightShift on the energy proportionality curve, from Figure 1, with a 20% and 50% capable Knight. To



(a) 20% Capable

(b) 50% Capable

Figure 5: KnightShift enhanced energy proportionality curve

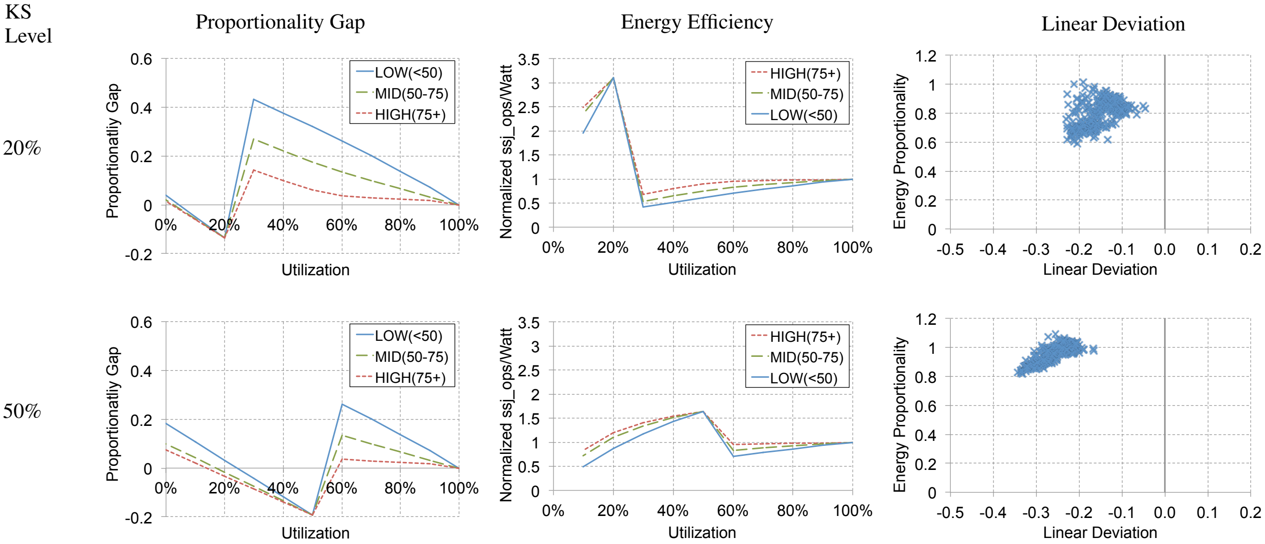


Figure 6: Effect of KnightShift on SPECpower commercial servers

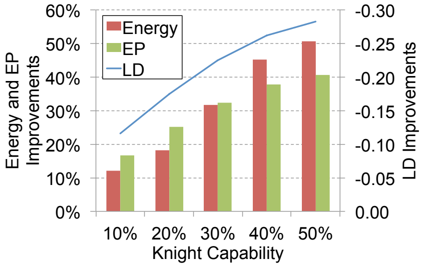


Figure 7: Server Energy, EP, LD improvement with KnightShift

generate this data, we assume that anytime the utilization is within the Knight's capability levels, the Knight will handle that requests. Otherwise, the primary server will handle the request. Note that in KnightShift, the Knight must remain on, which increases the peak power consumption of the server. The reason for this requirement was explained in the previous section. Even with the increase in peak power consumption, we still experience significant power savings because the servers spend the majority of the time in the low utilization regions. (more details will be presented in section 7). The primary server is shut down at low utilizations, allowing the Knight to handle all low utilization requests, significantly decreasing power consumption. Depending on the capability, energy savings vary. But in all cases, we shift the server to -LD domain, but with differing levels of -LD. It is interesting to note that at specific utilization levels, a KnightShift-enabled system can consume less power than an ideal energy proportional system, opening the possibility of servers operating with better efficiency than ideal energy proportionality. For instance, in Figure 5b when the server utilization is approximately between 20% and 50%, the overall power consumption is better than an ideal energy proportional server because the Knight uses less power than an ideal energy proportional server at that utilization.

Figure 6 shows the effect of KnightShift with a 20% and 50% capable Knight on proportionality gap, energy efficiency, and linear deviation (compare to Figure 2).

Proportionality Gap: At 20% capability, the proportionality gap of the KnightShift server is essentially eliminated at utilization below 20%. While in Knight mode, the proportionality gap is negative, meaning that the power used by the Knight at a specific utilization is lower than that of an ideal energy proportional server, as shown in Figure 5. At 50% capability, the proportionality gap is greatly reduced in the 0%-25% utilization range as compared to Figure 2d, while the proportionality gap is eliminated from 25%-50% utilization range. The reason for non-zero proportionality gap at the lower range is because of the power consumed by the Knight itself. As long as the proportionality gap exists at low utilization, KnightShift should benefit that server.

Energy Efficiency: The energy efficiency curves for the 20% and 50% Knight capabilities are shown in Figure 6. KnightShift enhances server's energy efficiency and allows them to run at or better than peak efficiency (great than 1 in the figure) even at lower utilization. The improvement is directly correlated with the reduction in proportionality gap. 20% capable Knights operate above peak efficiency between 0-25% utilization range. 50% capable Knights operate at above peak efficiency from 25%-50% utilization range, and

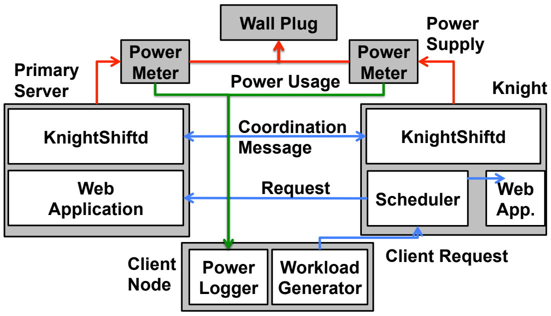


Figure 8: KnightShift Prototype setup

just below peak efficiency below 25% utilization. This data shows that KnightShift energy efficiency is substantially higher than the baseline shown in Figure 2e.

Linear Deviation: KnightShift effectively shifts all servers from +LD to -LD range. Improving LD is the only option to improve energy proportionality when dynamic range improvements are not feasible. With a 20% capable Knight, the lowest EP server amongst the 291 servers jumped from 20% to 60%. Thus KnightShift is able to improve the EP of servers by allowing commodity servers to exhibit -LD. For 50% capable Knights, we even see servers with EP > 1, indicating that KnightShift effectively closed the proportionality gap.

EP and Energy Savings: To evaluate energy savings, we assume server utilization distribution similar to Google datacenter servers in [6]. Figure 7 shows the average improvements to LD, EP and potential energy savings. For 20% Knights, we experience average EP improvements of 25%, average energy savings of 18% and average LD decreased by .175. As Knight capability increases, energy savings grows due to more opportunity to be in the Knight. For 50% capable Knights, we experience average energy savings of 51% and average EP improvement of 41%. By having KnightShift being configurable, vendors may pick a KnightShift implementation that is best suited for their performance and energy budget goals.

7. Evaluation

In this work we evaluate KnightShift using two approaches. First, we present a KnightShift prototype and run a real-world workload, WikiBench [20], to demonstrate feasibility and performance of KnightShift under realistic conditions. A prototype implementation, however, provides limited flexibility to change the hardware configuration parameters. Hence, we developed a queueing model based simulator that is validated against the prototype implementation. We then use the simulator to conduct a broad design space exploration using traces collected from USC's production datacenter.

7.1. Prototype Evaluation

7.1.1. Prototype Setup The KnightShift prototype is similar to Figure 3c. The exact experimental setup is shown in Figure 8 . In this setup the Knight is a Shuttle XS35 slim PC with a 1.8GHz Intel Atom D525, 1GB of ram, 500GB hard drive and operates from 15W-16.7W. At idle, the CPU and memory consumes 9W, with the disk and motherboard consuming 6W. The primary server is a Supermicro server with dual 2.13GHz 4-core Intel Xeon L5630, 36GB of ram, 500GB hard drive and consumes from 156W to 205W while active. Recall that we assume that it is reasonable for the Knight to have less memory than the primary server as the performance impact due to less memory is accounted for in the capacity measurement of the Knight. For our particular setup, our Knight is not capable of supporting memory size larger than 1GB.

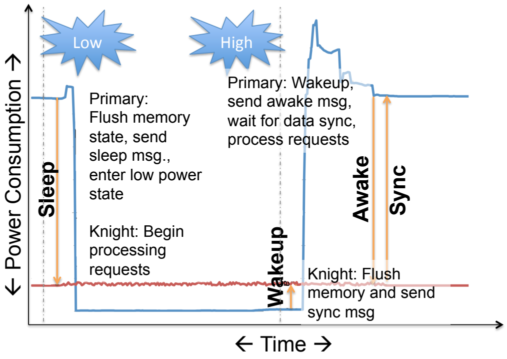


Figure 9: Coordination of KnightShift servers

Using SPECpower results we determined that our primary server has an EP of 24%. By enhancing the primary server with a Knight, we improved the EP to 48%! Note that although our primary server has a relatively low EP, if we use a significantly higher 70% EP server, our prototype Knight can still provide EP improvements. KnightShift is not meant to compete directly with servers that are already highly proportional across all utilization levels. KnightShift improves EP of servers that have large proportionality gap at low utilizations.

The primary server can turn on/off in 20/10 seconds, respectively. During Knight mode, the primary server is placed in hibernate mode, where the system is shutdown except for the network interface. It is also possible to place the primary server in suspend mode quicker than shutting down, but at the cost of higher idle power. Both nodes run Ubuntu Linux with all power-saving features enabled (DVFS, drive spin-down, etc). We determined that our Knight is 15% capable compared to the primary server using throughput measurements from apachebench [16]. In other words, the Knight's throughput is 15% of the peak throughput of the primary server. Request generation and power measurement data collection are handled by a separate client node which is not part of the prototype. The power consumption of the primary server and Knight are measured using two *Watts Up? Pro* power meters with data logged to the client node.

KnightShift Runtime: In our prototype setup, both nodes shares data through NFS, with the Knight acting as the NFS server. In order to force data consistency, we require coordination between both nodes. Thus, we require runtime software for KnightShift support. This software will handle both utilization monitoring and coordination. While there are many options for implementation, here we only present one particular implementation used in our prototype.

To support and enforce the KnightShift functionality, both nodes run a daemon, called KnightShiftd, to support utilization monitoring and coordination messages. KnightShiftd is implemented as a set of scripts and acts as the control center for the KnightShift system. KnightShiftd monitors the utilization of the node it's running on and makes mode switching decisions. To support redirection of requests, the Knight runs a scheduler, which acts as a simple load balancer to forward the requests to the active node.

Communication between both nodes takes place through messages. Figure 9 highlights the processes of switching between nodes, which also enforces data consistency. Upon entering a low utilization period, the KnightShiftd daemon will detect the low utilization of the primary server and initiates a mode switch. The primary server will flush its memory state to ensure that the latest data is up to date in the

	Response Time		Energy
	Average	95th	Consumption(KWH)
Prototype			
Baseline	144ms	249ms	23.27
KnightShift	150ms	296ms	15.35
Improvement	-4%	-19%	34%
Simulation			
Baseline	1.00	1.66	23.27
KnightShift	1.12	2.00	15.11
Improvement	-12%	-21%	35%
Error	8%	2%	1%

Table 2: Energy consumption and response time of Wikibench using our KnightShift prototype and simulator.

disk. When this completes, KnightShiftd will send a *sleep* message to the Knight and begin to power down. The KnightShiftd daemon on the Knight system receives the *sleep* message from the primary server, which is an indication that the Knight should begin processing requests. The Knight will process low utilization requests until it reaches a high utilization region. At this point, the daemon on the Knight will send a *Wakeup* message (through wake-on-lan) to wake up the primary server. When the primary server has booted up, the daemon on the primary server gets ready to process requests. It will send an *awake* message to the Knight. At this point, the Knight will flush its data and send a *sync* message, indicating to the primary server that it can resume processing requests.

7.1.2. Prototype Results To verify the correctness of KnightShift and to evaluate KnightShift under realistic workloads, we cloned Wikipedia and benchmarked it using real-world Wikipedia request traces. Wikipedia consists of two main components, Mediawiki, the software wiki package written in PHP, and a backend mySQL database. For our clone, we used a publicly available database dump from January 2008, containing over 7 million articles. We replayed a single-day Wikipedia access trace [32], which follows a diurnal sinusoidal pattern, using WikiBench [20], a Wikipedia based web application benchmark. Detailed WikiBench workload utilization profile for this case study is presented in [33].

The first three rows of data of Table 2 show the energy consumption and the 95th percentile response time of our KnightShift prototype compared to the baseline primary server. Service Level Agreements (SLA), which sets per-request latency targets, are typically based on 95th percentile latency [25].

We define the baseline as a system where all requests are always handled by the primary server. KnightShift is able to achieve 34% energy savings with only 19% impact on 95th percentile response time. This latency impact is mainly due to the single-threaded performance of the Knight rather than penalties due to switching between the Knight and primary server (Note that the average response time only increased by 4%). When running Wikibench only on the Atom-based Knight, we experience 95th percentile response time of 323ms for successfully completed requests. Thus, *KnightShift's 95th percentile response time is bounded by that of the Knight*. By using higher single-threaded performing processors, such as Intel Core i3, we should expect to experience response time bounded by the response time of the Core i3.

7.2. Trace-based Evaluation

7.2.1. Trace-based Setup While a prototype implementation provides great confidence regarding the functional viability and realistic improvement results, it also limits our ability to alter some of the critical design space parameters, such as Knight capability level, Knight performance, and Knight transition time. In order to fully explore these variables, we present KnightSim, a trace-driven KnightShift system simulator validated against our prototype system. During simulation runs, KnightSim replays the utilization traces collected from our production datacenter on a modeled KnightShift system. KnightShift is modeled as a G/G/k queue, where the arrival rate is time-varying based on the utilization trace, the service rate is exponential with a mean of 1 second, and varying k servers modeling the capability of the Knight and primary server. Because we do not have measured response time from out datacenter traces, we arbitrarily set the service rate to 1 second and report relative performance impact.

		Utilization		Δ Utililization	
Server	Type	$\bar{x}$	σ	$\bar{x}$	σ
aludra	stu. timeshare	3.87	3.12	0.59	0.84
email	email store	3.26	1.74	0.78	1.20
girtab	stu. timeshare	0.83	2.42	0.73	1.94
msg-mmp	email services	32.62	13.60	2.64	2.76
msg-mx	email services	19.23	7.41	1.69	2.30
msg-store	email store	11.05	5.88	2.39	2.72
nunki	stu. timeshare	4.86	10.85	1.98	4.50
scf	file server	5.47	4.19	1.15	1.65

Table 3: Datacenter trace workload characteristics

Datacenter Utilization Traces: In order to rigorously evaluate KnightShift under various workload patterns, we collected minute-granularity CPU and I/O utilization traces from our production datacenter over 9 days. The datacenter serves multiple tasks, such as e-mail store(email, msg-store1), e-mail services (msg-mmp, msg-mx), file server (scf), and student timeshare servers (aludra, nunki, girtab). Each task is assigned to a dedicated cluster, with the data spread across multiple servers. Selected servers within a cluster exhibit a behavior representative of each server within that cluster.

Table 3 shows the properties of each server workload along with its corresponding utilization and burstiness characteristics. Some of our servers (aludra, email, girtab, nunki, scf) run at less than 20% CPU utilization for nearly 90% of their total operational time [33]. These traces reaffirms prior studies that CPU utilization reaches neither 100% nor 0% for extended periods of time [6, 11, 27]. We also collected a second-granularity traces for a subset of these servers and found that there is a high correlation to minute-granularity. Thus, we use the minute-granularity data for the rest of the paper. The burstiness of the workload is characterized by $\sigma_{utilization}$, the standard deviation of the workload's utilization, and Δutilization, the change in utilization from sample to sample. $\sigma_{utilization}$ tells us how varied the utilization of the server is, while the Δutilization tells us how drastic the utilization changes from sample to sample. For example, nunki has a wide operating utilization range with large variation in utilization from sample to sample. More details of our datacenter traces are presented in [33].

Modeling Knight capability: Knight capability is modeled by varying the system capacity, k. For example, if we have a 10% Knight, then k = 10 in our G/G/k queueing model when operating in Knight mode. When the primary server becomes active then k = 100.

Scaling Power Consumption: To faithfully scale the power of the Knight as its capability changes, we assume simply that the power consumption of the CPU scales quadratically with performance. The quadratic assumption is based on historical data [5] which showed that power consumption increased in proportions to $performance^{1.7}$. We assume this is a reasonable assumption due to the fact that even if the Knight and primary server require similar infrastructure (such as same size memory), the Knight can tradeoff performance by using low-power components (such as low-power mobile memory), therefore, most components can scale.

Modeling Power: Our power model is based on our prototype system to allow us to compare and validate KnightSim. Through online instrumentation, we collect the utilization vs power data for both the Knight and primary server. We use this utilization-power data in our simulations; whenever a Knight is active at a given utilization we use the power consumption data collected from our prototype Knight. Similarly whenever the primary server is operating at a given utilization, we use the power consumption collected from the primary server in our prototype. It is also possible to generalize the power model and use a linear power model validated in [11].

In order to capture the energy penalty of transitioning to/from Knight, we conservatively model the transition power as a constant power during the entire wakeup period equal to the peak transition power. We determined empirically that the peak transition power for the primary server is 167W.

Arrival Rate and Latency Estimation: Our datacenter traces only have CPU and I/O utilization per second without individual request information. By assuming a mean service time of 1 second for each request, we can estimate a time-varying arrival rate through our utilization trace. For example, 50% utilization would correspond to an arrival rate of 50 requests per second. Through the simulated queueing model, we can obtain a relative average and 95th percentile latency of a KnightShift system compared to a baseline system.

Modeling Single-threaded Performance: We vary the queueing model's service time to model the performance difference of the Knight and primary server. We cannot infer single-threaded performance directly from processor frequency because single-threaded performance is based on frequency and the underlying architecture. Instead, we compare the 95th percentile latency of the Knight and primary server and scale the service time accordingly. For example, our primary server has tail latency of 249ms while our Knight has tail latency of 323ms as shown in section 7.1.2. As we do not have direct access to the datacenter servers, nor can we replicate the proprietary applications on our Knight, we cannot collect response times for the primary server and Knight for each individual workload. Therefore, in our model, we assume that all workloads experience similar performance slowdown due to the Knight similar to WikiBench, where the service time is increased by a factor of 1.3 compared to baseline.

Simulator Validation: We validated our trace-based emulation by collecting utilization traces from our WikiBench run and replayed the utilization traces through the trace emulator. In addition, we validated our power results against our prototype system by running a CPU and I/O load generator to match the utilization of the traces. Table 2 shows the results of our validation run. 95th percentile latency and energy consumption improvement results from KnightSim are all within 2% of our prototype system.

7.2.2. Sensitivity Analysis In this section we explore KnightShift's sensitivity to various parameters such as workload utilization patterns, Knight capability, and transition times.

Sensitivity to Workload patterns: We used KnightSim to simulate KnightShift running a variety of workload patterns by driving the queueing model with traffic patterns from Table 3. The energy and latency impact are shown in Table 4. Recall that our Atom-based Knight has a 95% response time that is 30% greater than the primary server, thus we consider any latency above 30% to be attributable to the KnightShift mechanism overhead. For workloads with low burstiness (aludra, email, msg-mmp, wikibench), we experience relatively low response time impact (<10%).

Trace	Energy Savings	95% Latency Impact
aludra	87.9%	40%
email	85.5%	37%
girtab	87.2%	49%
msg-mmp	-6.7%	7%
msg-mx	7.2%	254%
msg-store	34.5%	53%
nunki	67.7%	5989%
scf	77.5%	46%
wikibench	35.1%	21%

Table 4: Energy savings and latency impact wrt Baseline of a 15% Capable KnightShift system

For moderately bursty workloads (girtab, msg-store, scf), we experience latency impact within 25% of the Atom-based Knight. For these workloads, the majority of the latency impact occurs during the transition from the Knight to primary server when the Knight is handling requests that it cannot handle until the primary server is ready. These bursty behaviors tend to be periodic, thus it would be possible for KnightShift to learn day-to-day utilization patterns and proactively switch to the primary server to handle these high-utilization bursty periods, negating the high latency impact. This topic is outside the scope of the paper and will be explored in future work.

For very bursty workloads with high utilization (msg-mx, nunki), we experience the most latency impact, as expected. KnightShift does not handle scenarios where the workload switches quickly between very low and high utilization. In these scenarios, the workload may benefit from a higher capacity Knight.

Almost all workloads experience energy saving benefits from KnightShift with the exception of workloads with mostly high utilization periods. There are no benefits from using KnightShift for workloads that operate mostly at utilization above the capability of the Knight, hence such workloads don't need KnighShift support to begin with. For these cases, this may even lead to an energy penalty (msg-mmp) due to running the Knight alongside a heavily utilized primary server.

For most other workloads (aludra, email, girtab, scf, wikibench), we can experience an average of 75% energy savings with tail latency within 9% of the Atom-based server.

Sensitivity to Knight Capability: Figure 10 shows the effect of Knight capability levels on energy savings and 95th percentile response time. As Knight capability increases up to 50%, so does energy savings due to more opportunity for the system to stay in the Knight mode. Although the Knight uses more power at higher capability levels, increased energy savings from time spent in the Knight offset the Knight's higher power.

As Knight capability increases, up to a limit of around 50%, the primary server spends more time sleeping, resulting in latency converging to the 95th percentile latency of the Knight. At low Knight capability, especially for capability less than 20%, KnightShift thrashes; Knight cannot handle the tasks when switched to the Knight mode and these tasks endure long latency while waiting for the primary server to wakeup. Some workloads (msg-mx and nunki) experience latency penalties beginning at higher capability. These workloads do not experience latency impact at lower capabilities since KnightShift rarely switches to the Knight mode and hence the primary server handles nearly all the requests due to the high utilization demands. But when the Knight capability increases the system occasionally

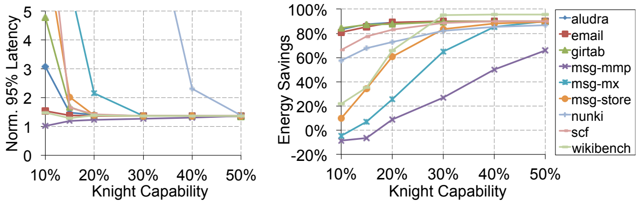


Figure 10: Effect of Capability on Latency and Energy

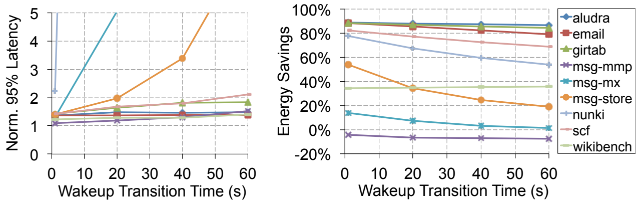


Figure 11: Effect of Wakeup transition time

switches to the Knight mode and the Knight is quickly saturated. Hence, the workload switches back to the primary server leading to latency penalties. KnightShift is in fact unnecessary for these workloads. Thus, for certain workloads with stringent QoS bounds KnightShift may not be an ideal solution.

Sensitivity to Transition Time: For brevity, we only present wakeup transition time. The effect of sleep transition time is similar. Figure 11 shows the effect of wakeup transition time on energy savings and 95th percentile response time. In general, as transition time increases, we experience less energy savings due to the primary server using power but not doing work, while the Knight is still handling requests it potentially cannot handle. This is reflected in an increase in 95th percentile latency as transition time increases.

Sensitivity to single-threaded performance: The tail latency of KnightShift is determined by the Knight. If the SLAs demand very tight latency slack (less than 20%), then it is best to use low-power processors, such as Core i3, as Knights instead of extremely low power Atom boards.

8. TCO

To study the effect of KnightShift on TCO of an entire datacenter, we use a publicly available cost model [17]. The model assumes an 8MW power budget where facility and IT capital costs are amortized over 15 and 3 years, respectively. The model breaks down TCO into server, networking, power distribution and cooling, power, and other infrastructure costs. We assume that KnightShift has no impact on rack density, with power budget as the sole limiting factor. In Table 5, we present our cost breakdown for our primary server and our Knight. We broke down cost into memory, storage, processor, and other system components. Other system components includes motherboard, chipset, network interface, fans, and other on-board components. A significant portion of the energy savings derive from other system components. This is due to the fact that many of these components are energy-disproportional, such as chipset, network interface, fans, and sensors. For example, the power consumption of motherboard components, such as chipset and network interface, are mostly constant with utilization. But with KnightShift, when we switch to the Knight, we could use a low-power mobile chipset (such as for Atom) rather than a higher power chipset (such as for Xeon) to save power. Performance is based on the SPECpower benchmark. An integrated version of KnightShift is expected to consume less power and have lower cost but we assume our prototype implementation of KnightShift to present worst-case TCO.

	Primary Server		Knight	
	Cost	Power(W)	Cost	Power(W)
Memory	$248	40	$20	3
Storage	$130	20	$70	18
Processor	$1102	70	$69	12
Other System Components	$350	75		
Total	$1830	205	$159	33
No. Servers	37361		34483	

Table 5: Cost breakdown of primary server and Knight based on prototype KnightShift system. Other system components include motherboard, chipset, network interface, fans, and other on-board components.

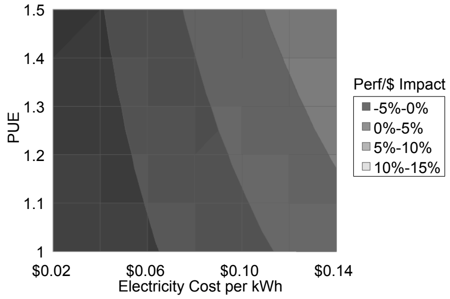


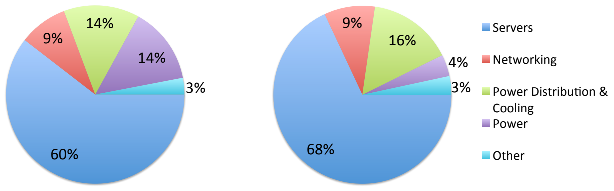
Figure 12: TCO breakdown across PUE and Energy Cost

We present TCO on a monthly basis as Performance per TCO Dollar spent (Perf/$), an important metric in TCO-conscious datacenters [22]. These results make the worst case assumption that both the Knight and primary server are always ON. Figure 12 shows the effect of PUE and electricity cost per kWh on Perf/$. There are two distinct regions, one where Perf/$ is improved, and one where Perf/$ is impacted. Due to the increased peak power usage of Knightshift and the fixed power budget of the datacenter, we suffer a decrease in total datacenter performance. Although there is a reduction in the number of servers due to peak power constraint, we do not always suffer any loss in Perf/$. In regions of higher electricity prices and higher PUE, it is easier to recoup the cost of KnightShift hardware due to more monetary savings per watt. For cases with high PUE and electricity cost, we experience up to 14% improvement to Perf/$. Only at very low electricity prices do we see a negative impact in Perf/$, due to the hardware cost outweighing the potential in energy savings. Note that even with PUE of 1, KnightShift can still provide Perf/$ advantages with electricity prices above $0.07 per kWh.

Figure 13 shows the TCO breakdown across server and infrastructure for PUE of 1.45 and electricity cost of $0.07. Although the total cost of servers is higher with KnightShift (68% total cost vs 60% in the baseline), the power budget improvements (from 14% to 4%), more than makes up for the difference, resulting in TCO savings of 11% monthly. Even by accounting for the lower number of servers, Perf/$/month improved by 4% compared to baseline.

9. Conclusion

Energy proportionality of computer systems has been increasing over the past few years. We introduce several metrics to analyze energy proportionality which shed light into why proportionality has not improved uniformly across all utilization levels. We show that servers exhibit significant proportionality gap at low utilizations. With the pervasiveness of multicores, servers in future will be rarely idle and hence energy saving techniques must now tackle the proportionality gap at low server utilization levels. We introduce KnightShift, a



(a) baseline (b) knightshift

Figure 13: TCO breakdown across servers and infrastructure

server-level heterogeneous architecture that fronts a primary server with a low-power compute node. By operating KnightShift at two levels of efficiency, we convert any server to exhibit sublinear energy proportionality, drastically improving energy proportionality. In our prototype KnightShift implementation with a 15% capable Atom-based Knight, we achieve a 2x improvement in energy proportionality (from 24% to 48%) due to improvements to both dynamic range and proportionality linearity. We demonstrated energy savings of 35% with latency bounded by the latency of the Knight using a real-world Wikipedia workload. In addition, we rigorously evaluated our prototype using various production datacenter traces and experience up to 75% energy savings with tail latency increase of about 9%. Through publicly available cost models, we also showed that KnightShift can improve performance per TCO dollar spent up to 14%. Our work hopes to motive future work in system-level active low-power modes that exploits low-utilization periods.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments. We also thank Sabyasachi Ghosh and Mark Redekopp for their early contributions which inspired this work. 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 *et al.*, "Somniloquy: augmenting network interfaces to reduce PC energy usage," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, Apr. 2009.

[2] H. Amur and K. Schwan, "Achieving power-efficiency in clusters without distributed file system complexity," in *ISCA'10: Proceedings of the 2010 International Conference conference on Computer Architecture*, Jun. 2010, pp. 222–232.

[3] V. Anagnostopoulou *et al.*, "Energy conservation in datacenters through cluster memory management and barely-alive memory servers." in *WEED '09: Workshop on Energy-Efficient Design*, 2009.

[4] D. G. Andersen *et al.*, "FAWN: a fast array of wimpy nodes," in *SOSP '09: Proceedings of the 22nd Symposium on Operating Systems Principles*, Oct. 2009.

[5] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's law through EPI throttling," in *ISCA'05: Proceedings of the 32nd international symposium on Computer Architecture*, 2005, pp. 298–309.

[6] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, dec 2007.

[7] J. S. Chase *et al.*, "Managing energy and server resources in hosting centers," in *SOSP '01: Proceedings of the 18th Symposium on Operating Systems Principles*, Dec. 2001.

[8] G. Chen *et al.*, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2008.

[9] B.-G. Chun *et al.*, "An energy case for hybrid datacenters," *SIGOPS Operating Systems Review*, vol. 44, no. 1, Mar. 2010.

[10] Q. Deng *et al.*, "MemScale: active low-power modes for main memory," in *ASPLOS '11: Proceedings of the 16th International Conference on Architectural support for programming languages and operating systems*, Mar. 2011.

[11] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA'07: Proceedings of the 34th international symposium on Computer architecture*, 2007, pp. 13–23.

[12] M. Ferdman *et al.*, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ASPLOS '12: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2012, pp. 37–48.

[13] S. Ghiasi, "Aide de camp: asymmetric multi-core design for dynamic thermal management," Ph.D. dissertation, 2004.

[14] E. Grochowski *et al.*, "Best of both latency and throughput," in *Proceedings of International Conference on Computer Design*, 2004, pp. 236–243.

[15] U. Hölzle, "Brawny cores still beat wimpy cores, most of the time." *IEEE Micro*, 2010.

[16] http://httpd.apache.org/docs/2.0/programs/ab.html, "ab - apache http server benchmarking tool."

[17] http://perspectives.mvdirona.com, "Cost of power in large-scale data centers."

[18] http://www.cpubenchmark.net/, "Passmark cpu benchmark."

[19] http://www.fit-pc.com/web/fit pc/, "fit-pc2."

[20] http://www.wikibench.eu, "Wikibench."

[21] R. Kumar *et al.*, "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," in *MICRO 36: Proceedings of the 36th International Symposium on Microarchitecture*, Dec. 2003, pp. 81–92.

[22] P. Lotfi-Kamran *et al.*, "Scale-out processors," in *ISCA '12: Proceedings of the 39th International Symposium on Computer Architecture*, Jun. 2012, pp. 500–511.

[23] K. T. Malladi *et al.*, "Towards energy-proportional datacenter memory with mobile DRAM," in *ISCA '12: Proceedings of the 39th International Symposium on Computer Architecture*, Jun. 2012, pp. 37–48.

[24] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *ASPLOS '09: Proceeding of the 14th International Conference on Architectural support for programming languages and operating systems*, Feb. 2009, pp. 205–216.

[25] D. Meisner *et al.*, "Power management of online data-intensive services," in *ISCA'11: Proceeding of the 38th international symposium on Computer architecture*, Jun. 2011, pp. 319–330.

[26] D. Meisner and T. F. Wenisch, "DreamWeaver: architectural support for deep sleep," in *ASPLOS '12: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2012, pp. 313–324.

[27] P. Ranganathan *et al.*, "Ensemble-level Power Management for Dense Blade Servers," in *ISCA '06: Proceedings of the 33rd international symposium on Computer Architecture*, Jun. 2006, pp. 66–77.

[28] V. J. Reddi *et al.*, "Web search using mobile cores: quantifying and mitigating the price of efficiency," in *ISCA'10: Proceedings of the 37th international symposium on Computer architecture*, Jun. 2010, pp. 314–325.

[29] F. Ryckbosch, S. Polfliet, and L. Eeckhout, "Trends in Server Energy Proportionality," *Computer*, vol. 44, no. 9, pp. 69–72, 2011.

[30] G. Semeraro *et al.*, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *MICRO 35: Proceedings of the 35th international symposium on Microarchitecture*, Nov. 2002.

[31] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *SIGMOD '10: Proceedings of the 2010 International Conference on Management of Data*, Jun. 2010.

[32] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 53, no. 11, Jul. 2009.

[33] D. Wong and M. Annavaram, "Evaluating a prototype knightshift-enabled server," in *WEED '12: Workshop on Energy-Efficient Design*, 2012.

[34] D. Wong and M. Annavaram, "Scalable System-level Active Low-Power Mode with Bounded Latency," University of Southern California, Tech. Rep. CENG-2012-5, 2012.

[35] www.spec.org/power_ssj2008/, "Spec power_ssj2008."