Towards Energy Proportionality for Large-Scale Latency-Critical Workloads

David Lo[†], Liqun Cheng[‡], Rama Govindaraju[‡], Luiz André Barroso[‡] and Christos Kozyrakis[†]
Stanford University[†] Google, Inc.[‡]

Abstract

Reducing the energy footprint of warehouse-scale computer (WSC) systems is key to their affordability, yet difficult to achieve in practice. The lack of energy proportionality of typical WSC hardware and the fact that important workloads (such as search) require all servers to remain up regardless of traffic intensity renders existing power management techniques ineffective at reducing WSC energy use.

We present PEGASUS, a feedback-based controller that significantly improves the energy proportionality of WSC systems, as demonstrated by a real implementation in a Google search cluster. PEGASUS uses request latency statistics to dynamically adjust server power management limits in a fine-grain manner, running each server just fast enough to meet global service-level latency objectives. In large cluster experiments, PEGASUS reduces power consumption by up to 20%. We also estimate that a distributed version of PEGASUS can nearly double these savings.

1 Introduction

Warehouse-scale computer (WSC) systems support popular online services such as search, social networking, webmail, online maps, automatic translation, and software as a service (SaaS). We have come to expect that these services provide us with instantaneous, personalized, and contextual access to petabytes of data. To fulfill the high expectations for these services, a rapid rate of improvement in the capability of WSC systems is needed. As a result, current WSC systems include tens of thousands of multi-core servers and consume tens of megawatts of power [8].

The massive scale of WSC systems exacerbates the challenges of energy efficiency. Modern servers are not energy proportional: they operate at peak energy efficiency when they are fully utilized, but have much lower efficiencies at lower utilizations [7]. While the average utilization of WSC systems for continuous batch workloads can be around 75%, the average utilization of shared WSC systems that mix several types of workloads, including online services, may be between 10% and 50% [8]. At medium and low utilizations, the energy waste compared to an ideal, energy proportional system is significant. To improve the cost effectiveness of WSC systems, it is important to *improve their energy efficiency at low and moderate utilizations*.

For throughput-oriented, batch workloads such as analytics, there are several ways to improve energy efficiency at low utilization. We can consolidate the load onto a fraction of the servers, turning the rest of them off [44, 25]. Alternatively, we can temporarily delay tasks to create sufficiently long periods of idleness so that deep sleep modes are effective [28, 29]. Unfortunately, neither approach works for *on-line, data-intensive (OLDI) workloads* such as search, social networking, or SaaS. Such user-facing services are often scaled across thousands of servers and access distributed state stored across all servers. They also operate with strict service level objectives (SLOs) that are in the range of a few milliseconds or even hundreds of microseconds. OLDI workloads operate frequently at low or medium utilization due to diurnal patterns in user traffic [29]. Nevertheless, server consolidation is not possible, as the state does not fit in a small fraction of the servers and moving the state is expensive. Similarly, delaying tasks for hundreds of milliseconds for deep sleep modes to be practical would lead to unacceptable SLO violations. Even at very low utilization, OLDI servers receive thousands of requests per second. Finally, co-scheduling other workloads on the same servers to utilize spare processing capacity is often impractical as most of the memory is already reserved and interference can lead to unacceptable degradation on latency SLO [20, 27, 14, 46].

A promising method to improve energy efficiency for OLDI workloads at low or medium utilization is to *scale the servers' processing capacity to match the available work*. We can reduce power consumption by scaling voltage and clock frequency (DVFS) or the number of active cores in the server. Traditionally, DVFS is controlled by monitoring CPU utilization, raising the frequency when utilization is high and decreasing it when it is low. We demonstrate that DVFS schemes that use CPU utilization as the only control input are ineffective for OLDI workloads as they can cause SLO violations. We observe that OLDI workloads exhibit different latencies for the same CPU utilization at different CPU frequencies. This makes it particularly difficult to correlate CPU utilization to request latency at different DVFS settings and complicates the design of robust control schemes.

We advocate making OLDI workloads more energy efficient across the utilization spectrum through fine-grain management of the servers' processing capacity based on the observed, end-to-end, request latency. The proposed *iso-latency* power management technique *adjusts server performance so that the OLDI workload "barely meets" its SLO goal*. By directly using the overall request latency and SLO targets, instead of the inaccurate and indirect metric of CPU utilization, we can achieve robust performance (no degradation of the SLO), while saving significant amounts of power during low and medium utilization periods. We show that *iso-latency* power management achieves the highest savings when using a fine-grain CPU power control mechanism, Running Average Power Limit (RAPL) [5], which allows the enforcement of CPU power limits in increments of 0.125W.

The specific contributions of this work are the following. We characterize Google workloads, including search, with real-world anonymized traffic traces to show the high cost of the lack of energy proportionality in WSC systems. We also quantify why power management techniques for batch workloads or utilization-based DVFS management are ineffective for OLDI workloads. We show that latency slack is a better metric compared to CPU utilization when performing power saving decisions. Next, we characterize the potential of *iso-latency* power management using RAPL and show that it improves energy proportionality and can lead to overall power savings of 20% to 40% for low and medium utilization periods of OLDI workloads.

978-1-4799-4394-4/14/$31.00

Since the cluster is operated at low and medium utilizations for nearly half the time, *iso-latency* translates to significant power savings. The characterization of *iso-latency* demonstrates some non-intuitive results such as: a) aggressive power management works for tail latency and can be applied with minimal extra power at scale; b) ultra-low latency services benefit even more from aggressive power management techniques. Finally, we describe PEGASUS, a feedback-based controller that implements *iso-latency* power management. For search, PEGASUS leads to overall power savings of up to 20% in clusters with thousands of servers without impacting the latency SLO. In addition, PEGASUS makes clusters nearly energy proportional for OLDI workloads in the 20% to 50% utilization range. If we discount the idle power at zero utilization (due to power for fans, power supplies, DRAM refresh etc.), PEGASUS meets and often beats energy proportionality in this range. We also estimate that a distributed version of PEGASUS that identifies less utilized servers in the cluster to further reduce power on can achieve the full potential of *iso-latency* power management, or nearly 40% power savings. To the best of our knowledge, this is the first study to achieve such power savings on an actual warehouse-scale deployment and the first to evaluate fine-grained versus coarse-grained power management tied to application latency SLO metrics.

2 On-line, Data Intensive Workloads

2.1 Background

On-line, data intensive (OLDI) workloads such as search, social networking, and SaaS represent the vast majority of activity on the Internet. They are driven by user requests that require mining through massive datasets and then returning combined results from many servers in near real-time. Hence, they partition the data across a large number of servers and use multi-tier, high fan-out organizations to process each request. For instance, in web search, processing a user query involves fanning out the query through a large tree of nodes [12]. In addition, DRAM and Flash memory are often used for fast access to data.

OLDI workloads are quite different from large-scale batch workloads like MapReduce. In addition to high throughput requirements, OLDI workloads have strict requirements on end-user latency. In our search example, since the rendered content can only be sent back to the user after all steps have completed, a slowdown in any one server in the tree will lead to either a slowdown for the user or degraded results, either of which result in poor end-user experience. Social networking and SaaS exhibit similar behavior in terms of high fanout and a dependence on the slowest component. It is well understood that an additional server-side delay of just 400msec for page-rendering has a measurable impact on user experience and ad-revenue [40]. Moreover, the large number of users and requests makes it important to optimize tail latency, not just average latency. Achieving low tail latency at the scale and complexity of OLDI workloads is quite difficult [13], making power management challenging.

2.2 OLDI Workloads for This Study

We use two production Google services in this study.

search: We evaluate the query serving portion of a production web search service. *search* requires thousands of leaf nodes all running in parallel in order to meet the stringent SLO, which is on the order of tens of milliseconds. Figure 1 shows an example *search* topology: a user query is first processed by a front-end server, which then eventually fans out the query to a large number of leaf nodes [12]. The search index is sharded across all of the leaf nodes and each query is processed by every leaf. Relevant results from each leaf are then aggregated, scored, ordered, and sent to the front-end to be presented to the user. Most of the processing is in the leaf nodes that mine through their respective shards. Consequently, leaf nodes account for the vast majority of nodes in a search cluster and are responsible for an overwhelming fraction of power consumed. To drive *search*, we use an example serving index from Google. We use an anonymized trace of real user queries for load generation. We can also generate a controlled amount of load by replaying the query log at different QPS (queries per second) levels.

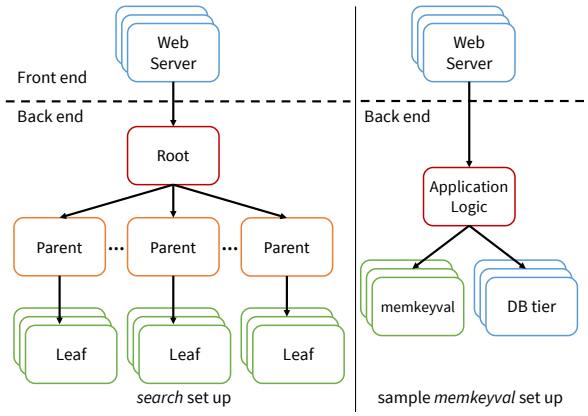

Figure 1. Configurations for *search* and *memkeyval*.

memkeyval: We also evaluate an in-memory key-value store used in production services at Google. Functionally, it is similar to *memcached*, an open source in-memory key-value store [3]. While *memkeyval* is not directly user-facing, it is used in the backends of several Google web services. Other large-scale OLDI services, such as Facebook and Twitter, use *memcached* extensively in user-facing services as well, making this an important workload. Figure 1 shows an example of how *memkeyval* is used by user-facing services. *memkeyval* is architected such that each *memkeyval* server is independent from all other *memkeyval* servers. Compared to *search*, there is significantly less processing per request and no direct fan out of requests. Instead, the fan-out occurs at a higher tier, where the web tier or application logic can issue tens or hundreds of requests to satisfy a single user request. Because of this, *memkeyval* has even tighter SLO latency constraints than *search*, typically on the order of tens or hundreds of microseconds. Hence, *memkeyval* adds to our analysis an example of an ultra-low latency workload. To generate load for *memkeyval*, we captured a trace of requests from production services, which we can replay at varying QPS speeds.

2.3 Power Management Challenges

Figure 2 shows the diurnal load variation for search on an example cluster. Load is normalized to the peak capacity of the cluster and power is normalized to the power consumed at peak capacity. There are several interesting observations. First, the cluster is not overprovisioned as it is highly utilized for roughly half the day, demonstrating that both the number and type of

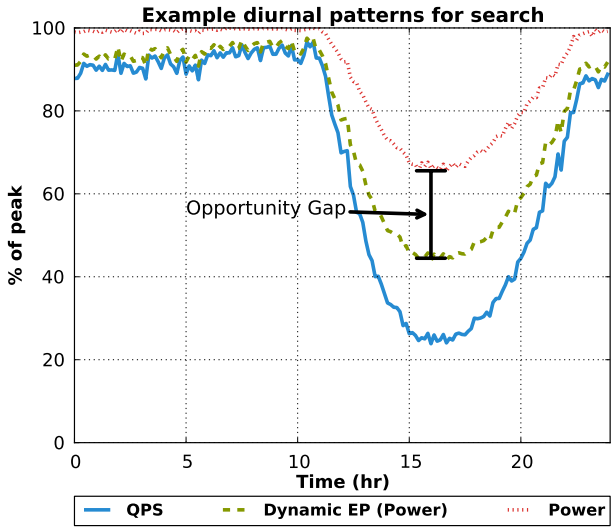

Figure 2. Example diurnal load and power draw for a search cluster over 24 hours. For a perfectly energy proportional cluster, power draw would perfectly track load (QPS). *Dynamic EP (Power)* **indicates the hypothetical amount of power consumed if the idle power is assumed to be constant and the active power is energy proportional.**

servers are well tuned for the peak of this workload. However, load is not constant. There is a very pronounced trough that represents a long period of low utilization. Second, the cluster exhibits poor energy proportionality during off-peak periods. Ideally, power consumption should closely track load. However, the actual power consumed is much higher: at 30% utilization the cluster draws 70% of its peak power. Since a major reason for lack of proportionality is the non-zero idle power (due to various components such as fans, power supplies, memory refresh, etc.), we also define a relaxed energy proportionality that only looks at the energy proportionality of dynamic power by discounting the idle power draw at 0% utilization. The *Dynamic EP* in Figure 2 is defined such that the power at 100% utilization is 100% of peak power, the power at 0% utilization is the idle power of the cluster, and all powers in between fall on a line connecting the two points. *Dynamic EP* represents the expected power draw of a cluster assuming that idle power cannot be removed and is often used to model server power [8]. Even after discounting idle power, the cluster draws far more power than expected at low utilizations. This is the opportunity gap we target: reducing the operating cost of large WSC systems by making them more energy proportional during periods of low or medium utilization.

Existing power management techniques for WSC systems cannot address this opportunity gap for OLDI workloads. Several techniques advocate consolidating workloads onto a fraction of the available servers during periods of low utilization so that the remaining servers can be turned off [20]. Hence, while each server is not energy proportional, the WSC system as a whole becomes nearly energy proportional. Nevertheless, workload consolidation is difficult for OLDI workloads as the number of servers is set by both peak processing requirements **and** data storage requirements. For instance, a search cluster is sized to hold the entire search index in memory, and individual nodes cannot be powered off without losing part of the search index, which will impact search quality. Even if spare memory storage is available, moving tens of gigabytes of state in and out of servers is expensive and time consuming, making it difficult to react to fast or small changes in load. Other techniques focus on individual servers and attempt to maximize idle periods so that deep sleep power modes can be used. For instance, PowerNap would need to batch tasks on each server in order to create sufficiently long idle periods [28, 29]. Full system idle power modes, such as ACPI S3, have transition times on the order of seconds [29]. Since OLDI workloads have latency requirements in the order of milliseconds or microseconds, use of idle power modes produces frequent SLO violations [29]. Moreover, OLDI workloads usually have a large fan-out (see Figure 1) when processing user queries. A single user query to the front-end will lead to forwarding of the query to all leaf nodes. As a result, even a small request rate can create a non-trivial amount of work on each server. For instance, consider a cluster sized to handle a peak load of 10,000 queries per second (QPS) using 1000 servers. Even at 10% load, each of the 1000 nodes are seeing on average one query per millisecond. There is simply not enough idleness to invoke some of the more effective low power modes.

3 Energy Proportionality Analysis for OLDI

3.1 Methodology

To understand the challenges of improving energy efficiency for OLDI workloads during low utilization periods, we collected power and latency measurements from a cluster with Sandy Bridge CPUs running production workloads. The power measured is full system power of the cluster and is equivalent to the power draw at the wall socket. For *search*, latency was measured at the root of the fan-out tree as opposed to the latency at each individual leaf node. By measuring as close to the end-user as possible, we have a more accurate assessment of the latency of the entire cluster. For *search*, we use the 30 second moving average latency (μ/30s) as one of the latency SLO metrics. Since we measure latency at the root where the effects of slow leafs are already seen, using mean latency at the root is an acceptable metric. We also collected 95%-ile and 99.9%-ile tail latencies at the root for *search* as a sensitivity analysis for further insights. For *memkeyval*, we measure mean latency, as well as 95%-ile and 99%-ile tail latencies. This is because *memkeyval* is typically used in such a way that the higher-level service waits for several parallel *memkeyval* requests to several servers to complete before being able to return a result. Thus, the overall latency of the entire *memkeyval* service is dependent on the slowest server, which is estimated by the tail latency of an individual server.

3.2 Analysis of Energy Inefficiencies

Figures 3 and Figure 4 show the total cluster power and the CPU-only power respectively at varying loads for *search* and *memkeyval*. As expected, the total cluster power consumed is far higher than the power consumed by an ideal, energy proportional cluster. We also plot the power draw of a cluster that exhibits energy proportional behavior on its active power (*Dynamic EP*). Eliminating idle draw due to fans, power supplies, and leakage power is quite difficult in the absence of sufficiently long idle periods. However, we expect that the cluster should be fairly close to energy proportional once idle power is removed. Figure 3 shows this is not the case in practice. The total cluster power con-

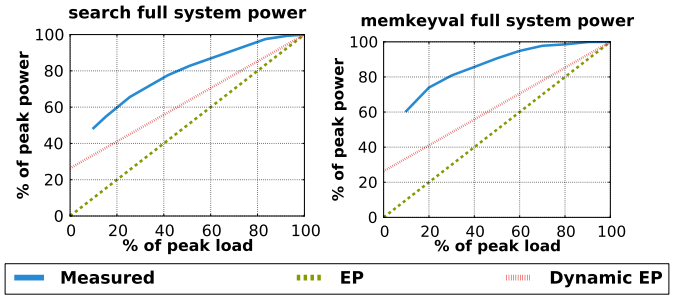

Figure 3. Total cluster power for *search* and *memkeyval* at various loads, normalized to peak power at 100% load.

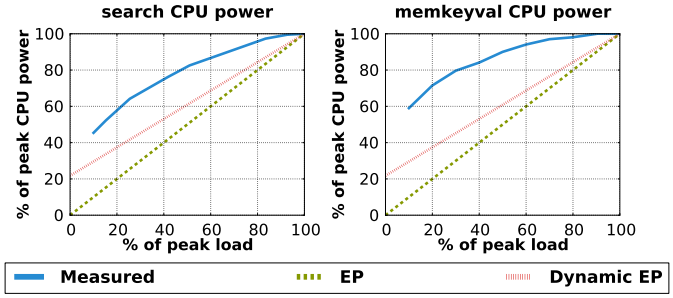

Figure 4. CPU power for *search* and *memkeyval* at various loads, normalized to peak CPU power at 100% load.

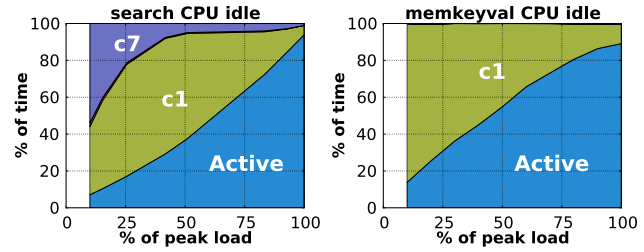

Figure 5. Characterization of CPU sleep states for *search* and *memkeyval* at various utilizations.

sumed is still much higher than the *Dynamic EP power* across all utilization points for both workloads. While idle power is a major contributor to the energy proportionality problem, it alone does not account for the large gap between the power behavior of current clusters and the behavior of energy proportional clusters. To improve energy proportionality, we focus on addressing active power draw when the cluster is underutilized. Improving idle power is important but outside the scope of our work.

Since the CPUs are the largest contributors to server power in modern servers (40% of total power at idle and 66% at peak [8]), we focus on CPU power as measured directly using on-board sensors in the servers (Figure 4). The power consumed by the CPU is not energy proportional and has the same shape as the power curves of the entire cluster (Figure 3). This establishes that the CPU is a major contributor to the non-energy proportionality for these OLDI workloads, even if idle power is ignored. Therefore, we focus on understanding the cause of non-energy proportionality on the CPU and how to reduce the power consumed by the CPU in the context of making the entire WSC system more energy proportional.

The default CPU power management approach for the servers we study is to execute at the highest available CPU active power

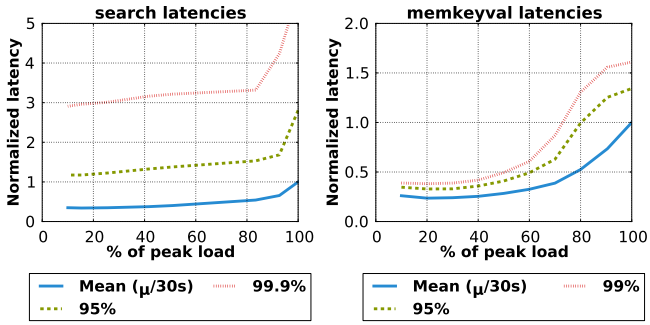

Figure 6. Characterization of latencies of *search* and *memkeyval* at various utilizations. The latency for each service is normalized to the average latency of the service at max load.

state (p-state), followed by idling in some sleep state. Namely, the power saving driver in use is *intel_idle* [1]. We characterized the fraction of time each core spends active and the fraction of time it spends in various sleep states (c1, c3, c7) in Figure 5. In c1, the core is clock gated. In c3, the L1 and L2 caches are flushed and clock gated. In c7, the core is power gated [37]. c1, c3, and c7 are ordered in increasing order of power savings [4]. c1, c3, and c7 have wake up latencies of almost zero, 10μs, and 100μs [4, 29]. We observe that both *search* and *memkeyval* do not spend very much time in the low power c7 sleep state. The vast majority of idle time is spent in the c1 sleep state, which does not save very much power but has almost zero exit latency. This is because in OLDI workloads, idle periods are particularly short even at low utilization, providing little opportunity to exploit deeper sleep states with higher exit latencies [29]. The idle behavior of OLDI workloads contrasts sharply with application behavior for mobile and client devices, where racing to idle is effective because of long idle times between short bursts of activity [33]. These observations motivate us to improve proportionality by improving active mode power consumption and by reducing the (ineffective) CPU idle time for OLDI workloads.

Figure 6 provides a clue towards achieving this goal. It shows how the various latencies of *search* and *memkeyval* are increasing with the amount of load on the cluster. It is important to remember that SLO targets for user or internal requests are set at a fixed value. These targets are independent of the immediate load on the cluster. Furthermore, SLO targets are typically set right at the knee of the latency/throughput curve. At loads below these inflection points, we observe that: a) there is still significant headroom between the measured latency and the SLO target; b) workload developers and system operators only care about avoiding SLO latency violations. This stems from the fact that SLO targets represent performance that is acceptable to the operator. The gap between the measured latency and the SLO target suggests that there are energy inefficiencies caused by the cluster unnecessarily overachieving on its SLO latency targets. We exploit this observation to reduce power: we slow down the CPU as much as possible under the current load in order to save power without violating the overall workload SLO target.

3.3 Shortcomings of Current DVFS Schemes

It is important to understand why existing power management systems based on Dynamic Voltage Frequency Scaling (DVFS), such as Linux's built-in *cpufreq* driver, cannot be used exploit

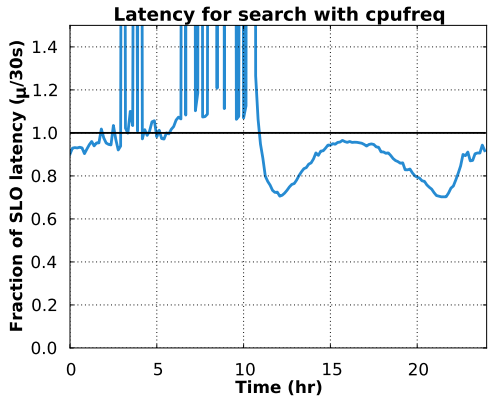

Figure 7. Latency of *search* with *cpufreq*, showing SLA violations. The solid horizontal line shows the target SLO latency.

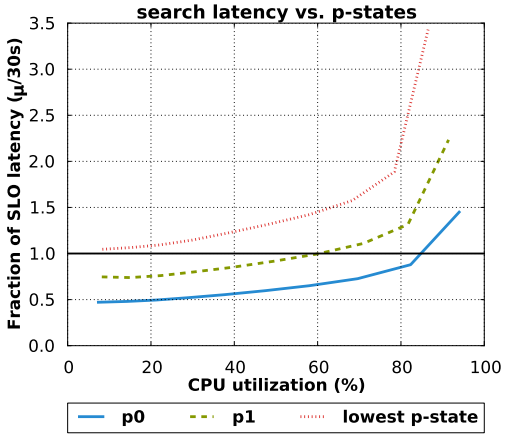

Figure 8. Characterization of *search*'s latency dependency on both p-state and CPU utilization. The solid horizontal line shows the target SLO latency.

the latency slack at medium and low utilization in order to save power for OLDI workloads. These systems operate by keeping the CPU utilization within a fixed range by changing the CPU's operating frequency through p-states. For example, the *conservative* governor in Linux's *cpufreq* driver keeps the CPU utilization within 20% to 95%. When CPU utilization goes below/above the low/high threshold for a certain amount of time, the driver will transition to the next lower/higher p-state. However, this approach leads to frequent latency spikes for OLDI workloads. We ran *search* with the 24-hour trace with *cpufreq* enabled. Figure 7 shows significant latency spikes and SLO violations as *cpufreq* fails to transition to the proper p-state in several cases. This is the reason that the baseline power management policy in the search cluster does not use *cpufreq* and DVFS control.

We demonstrate that the inability of *cpufreq* to lower power without causing SLO violations is because CPU utilization is a poor choice of control input for latency critical workloads. Figure 8 shows that the latency of *search* is different at the same CPU utilization across different p-states. Therefore, CPU utilization alone is fundamentally insufficient to estimate latency. No amount of tuning CPU utilization thresholds would allow a DVFS controller to achieve optimal power savings without SLO violations. Depending on the workload complexity and its SLO targets, the thresholds will be different and would vary for each p-state. Thus, we argue that to save power without SLO violations, we need to directly use application-level latency as the input to power control. Note that the use of CPU utilization as the only control input is a shortcoming not only limited to *cpufreq*: several research proposals [15, 17, 31, 35], Intel's demand-based switching mechanism [11], and most OS implementations also use CPU utilization as the sole control input.

Another shortcoming of existing DVFS systems is that they use p-states to manage the tradeoff between power and performance. However, p-state based power management is extremely coarse grained. For instance, dropping from p0 to p1 would disable TurboBoost, which can result in a significant frequency drop, up to 700MHz for an example Xeon E5-2667v2 CPU [2]. This means that using p-states alone misses on opportunities for power savings. For instance, in Figure 8, at utilization ranges between 65%-85%, the highest power p-state (p0) must be used, as the use of p1 would cause SLO violations. This motivates the need for fine-grain CPU power management in hardware.

4 Iso-latency Power Management

We propose a power management policy called *iso-latency* that addresses the shortcomings of existing DVFS systems. At a high level, it monitors the end-to-end request latencies of an OLDI service and adjusts the power settings of all servers so that SLO targets are barely met under any load. *Iso-latency* is inherently a dynamic policy as there is no one CPU p-state that will give optimal power savings while preventing SLO violations for all loads and all workloads. Intuitively, *iso-latency* is making a trade-off between latency and power. Figure 6 shows that, at low and medium utilization, there is latency headroom that can be exchanged for power savings without impacting SLO targets.

4.1 Iso-latency Potential

Before we discuss our implementation of *iso-latency*, we analyze the potential benefits for our OLDI workloads and servers.

To adjust the power settings of servers, we use a fine-grain power management feature introduced in recent Intel chips named *Running Average Power Limit (RAPL)* [5]. The RAPL interface allows the user to set an average power limit that the CPU should never exceed. On the servers we evaluate, we used the default time period to average over (45ms). RAPL can be programmed dynamically by writing a model specific register (MSR), and changes to the power limit take effect within less than 1 millisecond. The interface exports a fine grained power control knob, as the power limit can be set in increments of 0.125W. RAPL enforces this power limit mainly by scaling the voltage and frequency but can also modulate the CPU frequency to achieve average frequencies in between p-state frequencies. Compared to p-states, RAPL allows us to exploit a far wider spectrum of power-performance tradeoffs, which is critical to determining the optimal operating point for *iso-latency*.

To understand the potential of *iso-latency* for *search* and *memkeyval*, we sweep the RAPL power limit at a given load to find the point of minimum cluster power that satisfies the SLO target. For sensitivity analysis, we use several SLO targets. The first SLO target is the latency of the workload at maximum utilization. This is a more relaxed constraint, as latency is high at peak throughput (past the knee of the latency-load curve). The second SLO target is more aggressive, as it is the latency mea-

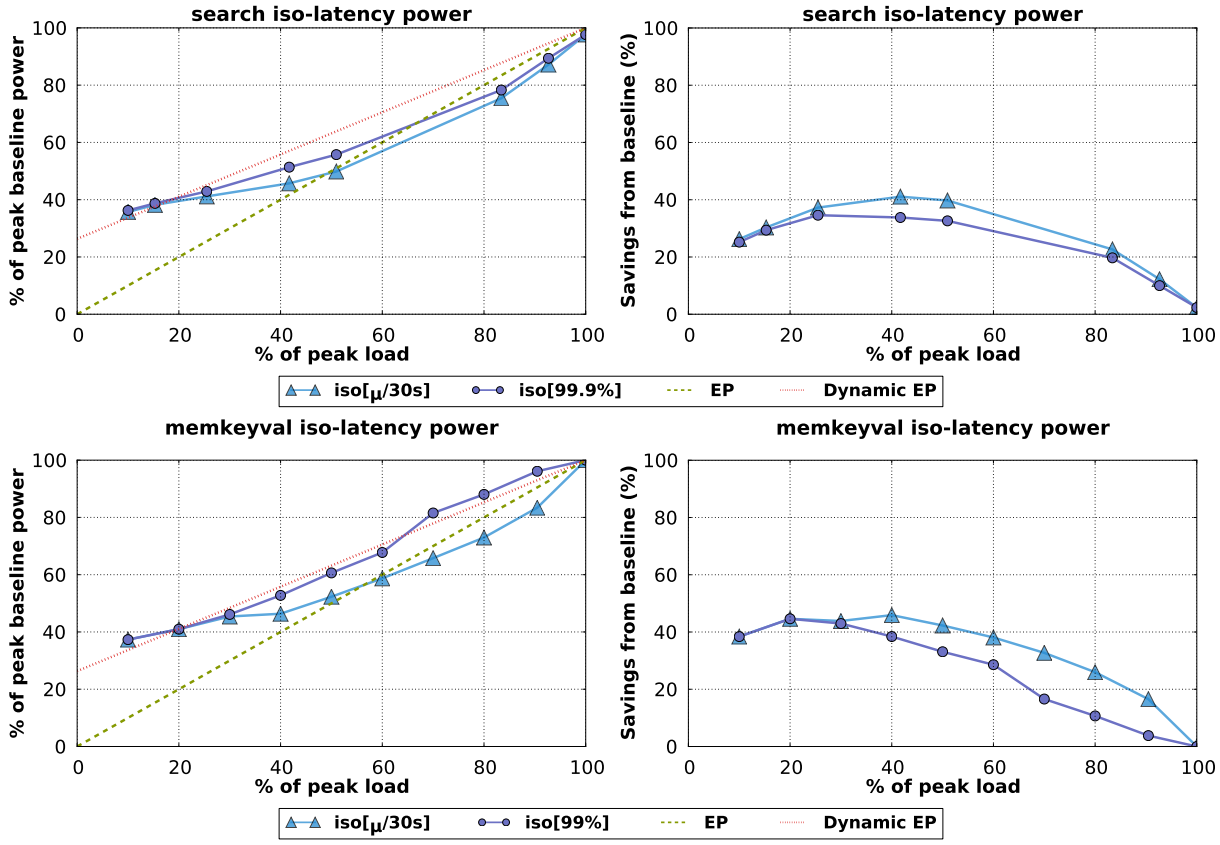

Figure 9. Characterization of power consumption for *search* and *memkeyval* for iso-latency with various latency SLO metrics. The SLO target is the latency at peak load (relaxed)

sured at the knee of the curve. For *search*, the knee is at 80% of peak utilization, while for *memkeyval*, the knee is at 60% of peak utilization. Figures 9 and 10 compare the baseline power consumption to that of a cluster managed by an ideal *iso-latency* controller that always finds the minimum power setting that avoids SLO violations. The right column of the figures shows the power savings using *iso-latency* over the baseline. For clarity, we omit the 95%-ile tail latency result for both *search* and *memkeyval*, as it is almost indistinguishable from the 99.9%-ile latency and 99%-ile latency curves for *search* and *memkeyval*, respectively.

The most salient observation is that *iso-latency* is able to save a significant amount of power compared to the baseline. In the utilization range of 20% to 80%, *iso-latency* can save 20% to 40% power and serve the same load **without** SLO violations. This result holds for both OLDI workloads, even for the aggressive SLO target (Figure 10), demonstrating that even a relatively small amount of latency headroom can be turned into large power savings. A physical analogy can be made to explain this result. The baseline can be compared to driving a car with sudden stops and starts. *iso-latency* would then be driving the car at a slower speed to avoid accelerating hard and braking hard. The second way of operating a car is much more fuel efficient than the first, which is akin to the results we have observed.

Another interesting observation from Figures 9 and 10 is that *iso-latency* can save significant amounts of power even when the SLO target is enforced on tail latency instead of on mean latency. For both *search* and *memkeyval*, *iso-latency* does not need to spend too much additional power to meet tail latency constraints, such as low 99.9%-ile latency. This is because most of the latency is due to queueing delay, which is responsible for the sharp knee in the latency curves [13]. *Iso-latency* needs to run the cluster just fast enough to avoid excessive queueing delays. While it does take a little more power to avoid queueing delay effects in tail latency, it is not significantly more power than to avoid excessive increases in mean latency.

It is important to note that *iso-latency* compares well to idealized, energy proportional clusters. For the relaxed SLO target (Figure 9), *iso-latency* leads to lower power consumption than an energy proportional (EP) cluster at utilizations higher than 50%. The explanation is fairly simple: by using DFVS to reduce the CPU speed, *iso-latency* achieves better than linear reductions in power ($P = CV^2F$) by reducing F almost linearly and V by significant amounts as well. At utilizations lower than 50%, the ideal EP cluster does better than *iso-latency* because the total power consumed is dominated by idle power. When idle power is removed (Dynamic EP), *iso-latency* performs as well as, if not better, than Dynamic EP at all utilizations. When we consider aggressive SLO targets (Figure 10), *iso-latency* matches the ideal EP curve above 60% utilization and outperforms the Dynamic EP curve above 30% utilization. A tighter SLO latency causes the power curves to be shifted up; nevertheless, *iso-latency* still saves a significant amount of power over the baseline.

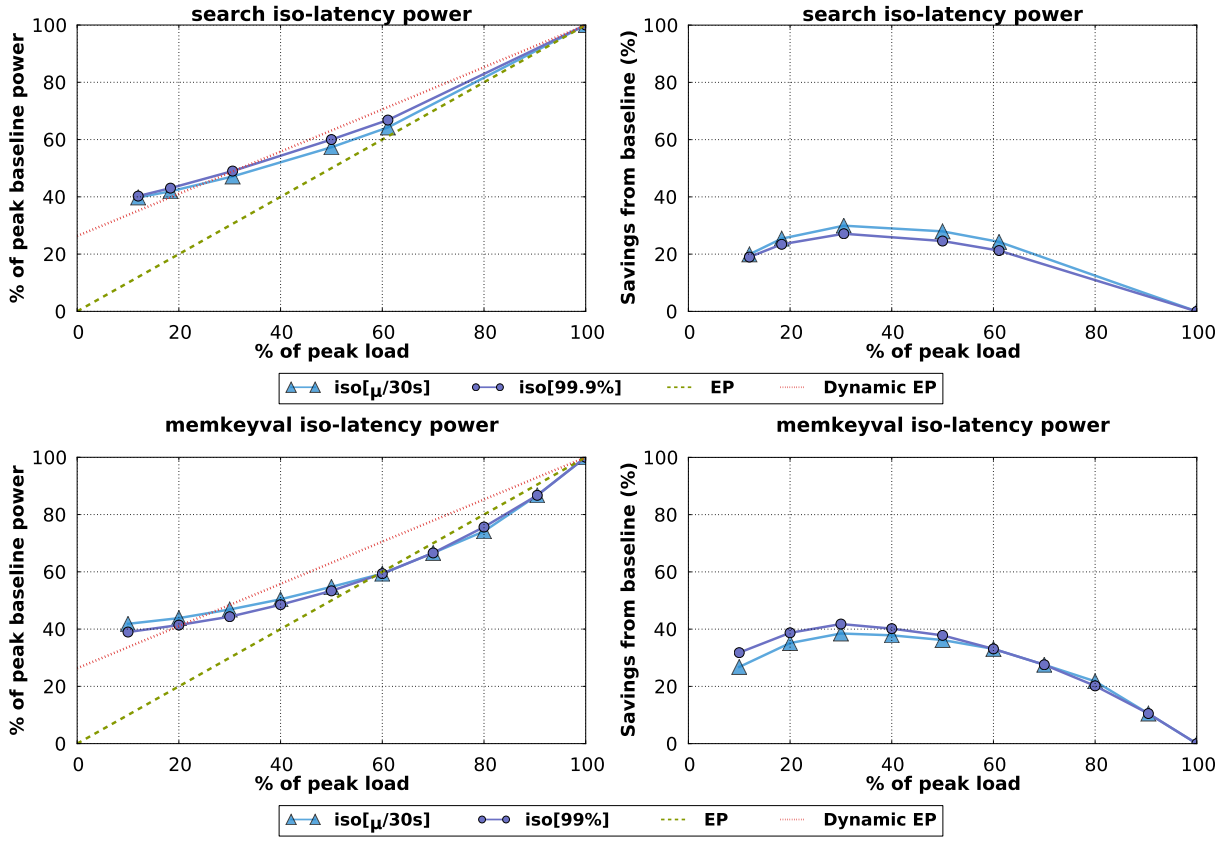

Figure 10. Characterization of power consumption for *search* and *memkeyval* for iso-latency with various latency SLO metrics. The SLO target used is more aggressive compared to Figure 9.

Iso-latency leads to higher power savings for *memkeyval* compared to *search*, even though the former has tighter absolute latency constraints (μsec compared to msec). Recall the characterization of idle times from Figure 5. We see that *memkeyval* cannot take advantage of deeper sleep states and can only use c1 because of the high request arrival rate, even at 10% load. On the other hand, because *search* has a lower request arrival rate, the baseline can take advantage of more power efficient sleep states. Thus, we expect *iso-latency* to benefit ultra-low-latency, high throughput workloads more, because it can convert the "race to (inefficient) idle" scenario to a more energy efficient "operate at a moderate but constant rate".

4.2 Using Other Power Management Schemes

As mentioned earlier, we can use the *iso-latency* policy with other power management mechanisms such as p-states. While we believe that RAPL is a better management mechanism because it allows for fine-grain control, it is not currently available on all CPU chips. Another power management mechanism is to consolidate the workload onto a sufficient subset of the cores in each CPU during times of low utilization. The remaining cores can be powered off (i.e., put into c7 sleep state) so that power is saved without experiencing SLO violations. We characterize *iso-latency* with CPU consolidation in Figure 11. Instead of sweeping RAPL settings to find the most appropriate for each utilization level, we now sweep the number of active cores per server. We leave the CPU operating at its highest p-state in order

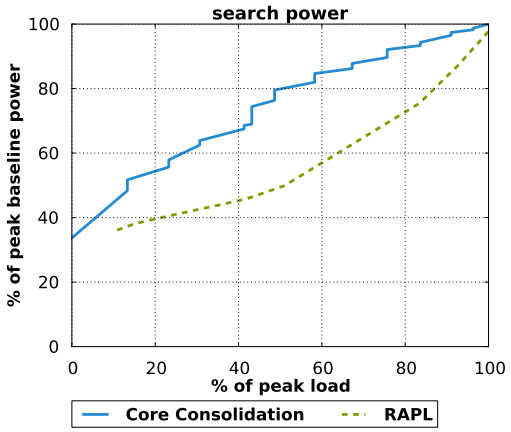

Figure 11. Comparison of iso-latency with core consolidation to RAPL. Measured total cluster power consumed by *search* at various utilizations, normalized to peak power at 100% load.

to maximize the number of cores that can be turned off. The results in Figure 11 are for *search* using the relaxed SLO target on the mean latency. These are the easiest constraints one can put on *iso-latency* to obtain maximum power savings.

Figure 11 shows that core consolidation is not as effective as RAPL when used as a power management scheme for *iso-latency*. The static power savings achieved by shutting down cores are eclipsed by the far larger dynamic power required to

operate the remaining cores at a high frequency. This is the basic phenomenon driving the multi-core evolution: assuming sufficient parallelism, many slow cores is more energy efficient than fewer faster cores. Nevertheless, it is still interesting to explore how core consolidation can be combined with RAPL (DVFS control) to create a better power management mechanism. Depending on the exact tradeoffs between static and dynamic power at low frequencies, it may be better to have a few moderately fast cores as opposed to many slow cores. This is an interesting tradeoff to examine in future work.

5 A Dynamic Controller for Iso-latency

Having established the potential of *iso-latency* as a power management policy, we now discuss the implementation of a cluster-wide, *iso-latency* controller for OLDI workloads.

5.1 PEGASUS Description

PEGASUS (**P**ower and **E**nergy **G**ains **A**utomatically **S**aved from **U**nderutilized **S**ystems) is a dynamic, feedback-based controller that enforces the *iso-latency* policy. At a high level, PEGASUS is a feedback-based controller that uses both the measured latency from the OLDI workload and the SLO target to adjust the RAPL settings in a fine-grain manner across all servers. During periods with significant latency headroom, PEGASUS will automatically lower the power limit until the latency headroom has been reduced. Conversely, when measured latency is getting close to the SLO target, PEGASUS will raise the power limit until there is sufficient latency headroom. PEGASUS always keeps a small latency headroom in order to ensure stability and robust handling of load increases. This latency headroom is configurable and determined based on the gradient of load spikes. Our current implementation of PEGASUS assumes a uniform cluster and thus applies the same RAPL setting across all servers. We discuss the implications of this assumption in Section 5.2.2.

Figure 12 shows a diagram of PEGASUS. The *Application Performance Monitor* monitors workload latency and reports the amount of headroom to the *PEGASUS controller*. This component can piggyback on performance monitoring infrastructure already in OLDI workloads. The *PEGASUS controller* in turn is responsible for determining the proper power adjustment based on the latency headroom, which it then periodically communicates to each of the *local agents* running on each server. The power adjustment is informed by the power savings policy, which is tailored to the OLDI workload. The frequency at which the *PEGASUS controller* sends updated power limits to the *local agent* is determined by how long it takes before the effects of the new power limit are seen in the application latency. This delay is application dependent, due to many workload specific factors such as the inertia of queries already in flight in the cluster, the delay in the latency measurement system, etc. The *PEGASUS controller* will send updates at a fixed frequency to the *local agents*, even if the power limit is staying constant. Finally, the *local agents* set the power limit from the *PEGASUS controller* using the RAPL interface. In case of a communication breakdown between the *PEGASUS controller* and the *local agents*, the *local agents* will automatically default to maximum power. Hence, PEGASUS failures cannot jeopardize the SLO for the workload.

The *PEGASUS controller* is very simple, around 200 lines of code, and has modest resource requirements. In its current

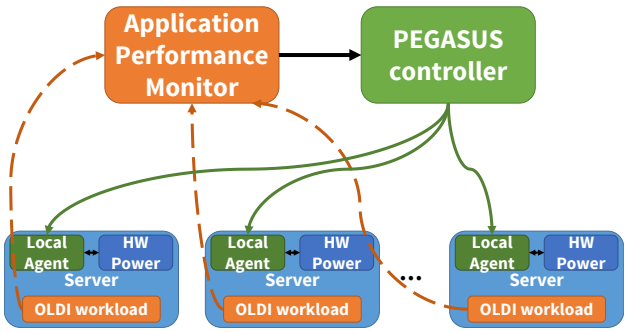

Figure 12. Block diagram showing high level operation and communication paths for PEGASUS.

form, it is based on a multi-step bang-bang controller that outputs a $\Delta(PowerLimit\%)$ depending on the latency headroom. The thresholds and the control outputs are workload dependent and are determined empirically. We describe control parameters for *search* in Table 1 For our experiments, we ran the *PEGASUS controller* on a single core of a single server. In our experiments, running on one core is more than enough to control the thousands of servers in the cluster. We profiled the performance of the *PEGASUS controller*, and determined that most of the time ($> 90\%$) is spent on transmitting the network packets to the *local agents*. When scaled to thousands of servers, the *PEGASUS controller* takes approximately 500 milliseconds to send out the command packet to all of the *local agents*. This delay can be shortened by parallelizing the networking portion of the *PEGASUS controller* to use multiple cores. However, for our experiments, the 500 millisecond delay does not appear to negatively impact the control scheme nor the performance of the OLDI workload.

We designed PEGASUS to be a general controller that works with any OLDI workload. PEGASUS is separated into two distinct components: policy (control parameters) and enforcement (actuation). The policy component is responsible for determining how much to adjust the CPU power limit. The policy portion is workload specific, and is dependent on the characteristics of the workload, such as the latency SLO metric (mean or tail latency) as well as the sensitivity of the workload to CPU power. For instance, *search* with its SLO target measured as the mean latency over 30 seconds will need to make smaller power adjustments compared to the same workload with its SLO target measured as the mean latency over 60 seconds. This is because PEGASUS needs to be more conservative when managing workloads with tighter SLO constraints, as there is less room for error. The enforcement portion is workload agnostic. It applies the CPU power limit determined by the policy of PEGASUS uniformly across all nodes in the cluster. Once the local agent receives a request from the central controller, it only takes milliseconds before the new power limit is enforced by RAPL.

5.2 PEGASUS Evaluation

We evaluate PEGASUS on the more challenging of the two OLDI workloads, *search*. We chose *search* because it is a complete, large-scale, OLDI workload with high fan-out across multiple tiers. In addition, because *search* is a complete service, measuring the overall latency of a user query is straightforward, as it is the latency at the root of the distribution tree. Moreover, the power savings potential for *search* is lower than that for *memkey-*

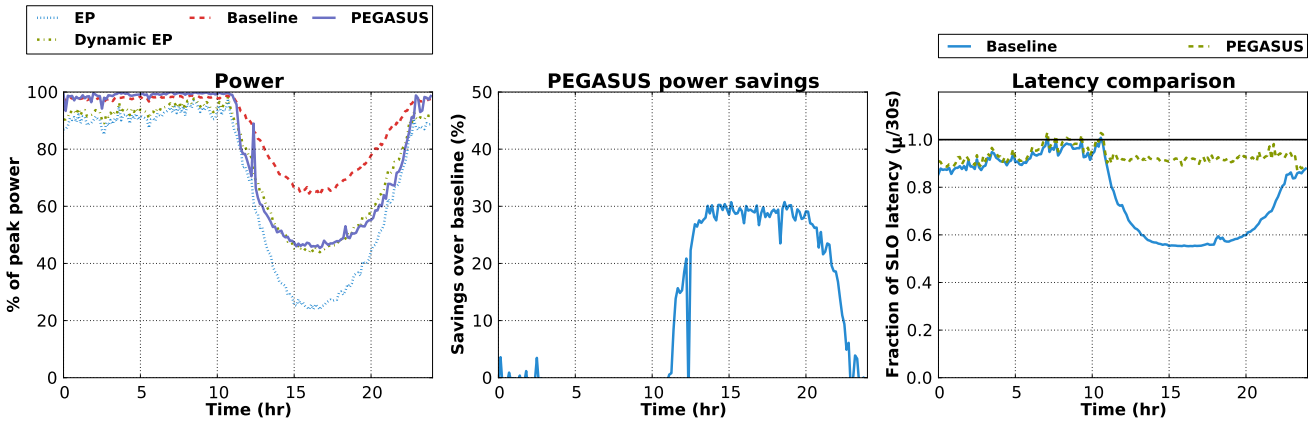

Figure 13. Results of running PEGASUS for *search* on the small cluster.

val (see Figure 10). Thus, evaluating PEGASUS on *search* paints a more conservative and realistic picture of its capabilities.

The evaluation setup closely approximates production clusters and their load for *search*. We use the same kind of servers and the same number of servers used for production websearch. We then populate these servers with portions of the Google search index. We generate search queries using an anonymized search query log containing searches from actual users. To evaluate a realistic mix of high and low utilization periods, we use a 24-hour load trace shown in Figure 2. Since we measure latency as close as possible to the user (root of the tree), we use the mean latency over a rolling 30-second window as the latency SLO metric. The SLO target is defined as the latency of *search* when operating at 90% of the maximum possible load supported by the cluster. To provide the necessary headroom for stability and load changes, we set the target latency for PEGASUS at 95% of the actual SLO target. For *search*, we only manage the leaf nodes, as they are by far the largest contributor to power consumption.

We describe the PEGASUS policy we used for *search*. It is a *very conservative* policy that aims to slowly reduce the power limit without causing any SLO violations along the way. The policy uses both the SLO latency (30 second moving average) and the instantaneous latency as inputs. The actions the policy takes based on the latency headroom is summarized in Table 1. The table lists rules in decreasing priority, i.e. only the first matching rule is applied. The rationale for using both SLO latency and instantaneous latency is that the instantaneous latency is a good early warning signal that lets PEGASUS know if an SLO violation might occur. This gives PEGASUS the opportunity to gracefully increase the power limit to stave off SLO violations before they happen. In addition, using the instantaneous latency allows PEGASUS to update the power limit more frequently, as opposed to waiting for the 30 second moving average to change. In our experiments, using the instantaneous latency allowed PEGASUS to update the power limit approximately once every 5 seconds. However, the measured SLO latency is still used as an emergency stop. If PEGASUS causes the SLO to be violated, then it should back off and not interfere with the OLDI workload. The reasoning behind this is that SLO violations are typically caused by excessive load and, in these situations, PEGASUS should not attempt to save power. When adjusting to save power, our policy for *search* takes very tiny steps to avoid riding the steep slope of the latency curve near the knee. Large perturbations near the knee can cause an order of magnitude increase in latency. Therefore, PEGASUS adopts a very cautious wait-and-see control scheme when the latency headroom is small. The policy also includes a built-in margin for the latency headroom where it will keep the current power setting. This margin is to account for noise in the latency measurement caused by variations in load as well as the differences between each search query.

Let $X = Measured\ SLO\ Latency$,
$Y = Measured\ Instantaneous\ Latency$, $T = SLO\ target$

Input	Action
$X > T$	Set max power, wait 5 minutes
$Y > 1.35T$	Set max power
$Y > T$	Increase power by 7%
$0.85T \leq Y \leq T$	Keep current power
$Y < 0.85T$	Lower power by 1%
$Y < 0.60T$	Lower power by 3%

Table 1. PEGASUS policy for *search*.

As before, the baseline point for comparison uses the default power manager that races to idle, *cpuidle*. We measure full system power for the entire cluster as well as query latency throughout the entire duration of the run.

5.2.1. Small Cluster Results: We first evaluate PEGASUS on an experimentation cluster that contains tens of servers. We scale the portion of the *search* index used to the memory capacity of the cluster. Because we are using a smaller dataset (index) and a smaller fan-out, the SLO target used in this experiment is a fraction of the SLO target of the production cluster.

Figure 13 shows the results of running the 24-hour trace described previously with and without PEGASUS. First, we note from the latency comparison graph that PEGASUS is able to meet the SLO target just as well as the baseline. In addition, the query latency with PEGASUS is flat, showing that PEGASUS is operating the cluster no faster than it deems necessary. Note that both the baseline and PEGASUS occasionally has small SLO violations (latency above 1.0). However, PEGASUS introduces no new SLO violations: it fails to meet the SLO in exactly the same cases as the baseline. These SLO violations are unavoidable as they are caused by load spikes exceeding the provisioned capacity of the cluster. Indeed, PEGASUS was not even attempting to save power at those points. Looking at the power graph for the first half of the experiment where SLO violations occur, the baseline and PEGASUS curves are identical. PEGASUS recognized

that there was not sufficient latency headroom to attempt power reduction measures.

The power savings graph in Figure 13 shows that PEGASUS achieves most of the theoretical potential of *iso-latency*. During the low utilization portion of the diurnal load pattern, PEGASUS saves 30% power compared to the baseline. This result establishes that the potential of *iso-latency* can be achieved with a relatively simple controller. Moreover, the power consumption graph shows that PEGASUS turns the cluster into an energy proportional one for *search*, if idle power is excluded. PEGASUS successfully trades off latency headroom for power savings. Over the entire 24 hour run, PEGASUS was able to save 11% total energy compared to the baseline. The overall energy savings are capped by the portion of time *search* operates close to peak capacity in the trace (roughly half of the time). Nevertheless, the power and energy savings from PEGASUS are essentially free, as these savings do not result in additional SLO violations.

5.2.2. Production Cluster Results: We also evaluated PEGASUS on a full scale, production cluster for *search* at Google, which includes thousands of servers. The salient differences between the full production cluster and the small cluster are that many more servers are available and that the entire search index is used. We also use the production level SLO latency target. For the production cluster experiments, we re-used the 24-hour trace discussed above in order to obtain an apples-to-apples comparison between the baseline and PEGASUS (two runs). We only use the 12 hours of the trace that contain the load trough and run it both with PEGASUS and the baseline approach because we are confident from the small cluster results that PEGASUS can successfully detect periods of peak utilization and operate the cluster at peak performance when needed. The 12 hours containing the trough characterize how effectively PEGASUS can turn the theoretical power savings of *iso-latency* into practical ones.

Figure 14 presents the results on the production cluster. The latency graph shows that PEGASUS did not cause any additional SLO violations compared to the baseline. This demonstrates that the enforcement portion of PEGASUS can scale out to thousands of nodes without issues. This result should not be surprising, because *search* must manage thousands of nodes as well. Distributing PEGASUS messages to thousands of servers is not that different from distributing queries to thousands of leafs. It can be done quickly in a coordinated manner. Compared with the traffic that *search* generates itself, PEGASUS messages to servers are rare and are a tiny fraction of overall traffic.

Nevertheless, we do observe several issues that only emerge at scale. The delay between iterations for PEGASUS needed to be increased (from 3 to 4 seconds) due to the increased inertia of queries within the cluster. Namely, when the power limit changed, the observed query latency evolved more slowly compared to that on the small cluster. Another issue is that PEGASUS is not able to save as much power on the full cluster as compared to the small cluster. In Figure 14, we observe power savings between 10% and 20% during the period of low utilization, compared to 30% in the small cluster. The reason is that PEGASUS applies uniform RAPL settings across all the servers with the assumption that all servers have the same CPU utilization. However, when scaling to thousands of nodes, this is no longer the case. We examined the distribution of CPU utilizations across the leaf servers and observed a wide variance across the nodes. For instance, when running at peak utilization, we observed that 50% of the nodes were operating at below 85% peak CPU utilization. At the same time, a tiny fraction (0.2%) of nodes were operating at 100% peak CPU utilization.

The existence of these hot nodes is due to the amount of work per query varying depending on the hotness of the index shard for the leaf node. Because *search* must wait for all leaf nodes to finish processing a single query before it can return results to the user, the entire cluster is bottlenecked by these hot nodes. The use of a single power limit forces all servers to run as fast as the small fraction of hot leaf nodes, significantly reducing the realized power savings. This is a challenge at large-scale that is not obvious from the small cluster results and demonstrates the difficulties in adapting a control scheme that works well for smaller clusters to a cluster with thousands of nodes.

The solution to the hot leaf problem is fairly straightforward: implement a distributed controller on each server that keeps the leaf latency at a certain latency goal. The latency goal would be communicated by the centralized controller that can monitor latency and SLO challenges at the root of the search tree. Using such a distributed approach, only the hot leafs would be run faster to allow the overall SLO target to be met, while the vast majority of leafs would be run at a much lower power. Compared to the simple, centralized version of PEGASUS, the distributed version would be more complicated to implement. However, distributed PEGASUS would be far more efficient at extracting power savings. It would also respond to latency transients faster. For instance, the centralized PEGASUS can only detect an impending latency problem when all of the queues in the search tree start to fill up, while distributed PEGASUS would be able to detect and act on individual queues filling up. Moreover, the distributed controller would be able to deal with heterogeneous server configurations in the cluster.

Unfortunately, due to scheduling constraints for the production cluster, we were unable to evaluate distributed PEGASUS in time for the paper submission. However, we estimated the power savings that can be achieved with this distributed controller. We measured the distribution of CPU utilization on the search cluster and combined it with the diurnal load curve to estimate how much power the distributed controller would be able to save. We show the results of our estimation in Figure 15. The distributed controller is estimated to perform much better than the uniform PEGASUS controller, saving up to 35% power over the baseline. This is comparable with the results from the small cluster, which shows that an idealized version of distributed PEGASUS can recover all of the power savings that were lost to using a uniform power limit. We also see that distributed PEGASUS has the opportunity to save power during periods of peak utilization, as it can save power on non-hot leaf nodes.

6 Related Work

There has been substantial work related to saving power for datacenter workloads. Ranganathan et. al propose a power budgeting approach that budgets for an "ensemble" of servers to take advantage of the rarity of concurrent load spikes in order to increase energy efficiency [36]. Raghavendra et. al propose a coordinated power management architecture for datacenters that integrates several otherwise disparate controllers together with-

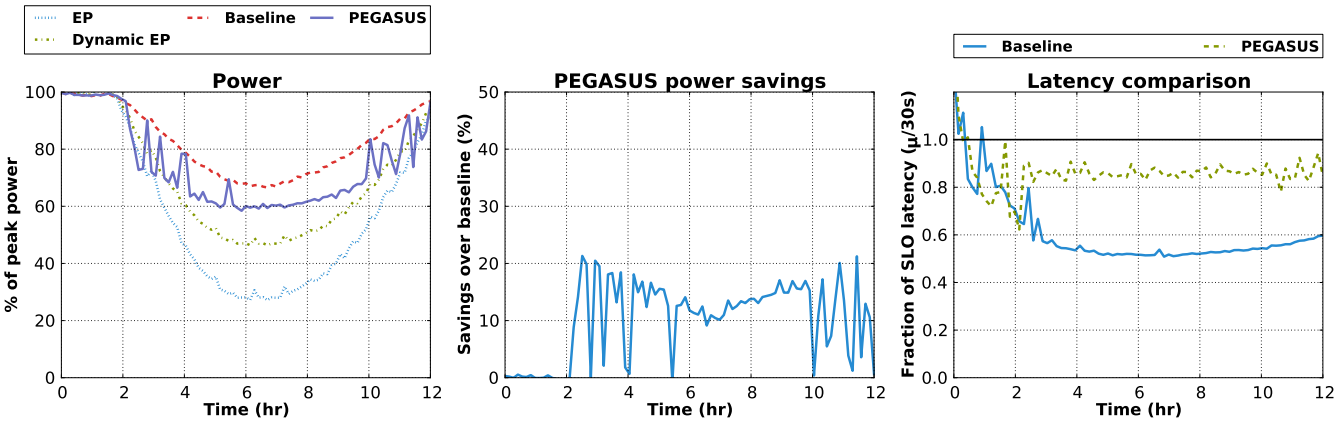

Figure 14. Results of running PEGASUS for *search* on the full production cluster.

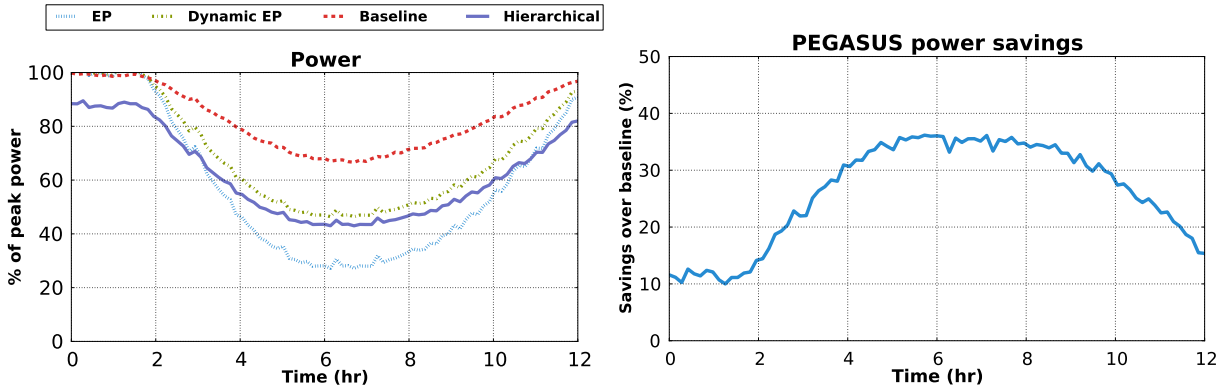

Figure 15. Estimation of power when using distributed PEGASUS.

out the different controllers interfering internally with each other [34]. PowerNap tackles the energy proportionality problem by having an enhanced "nap" state that is very low power and has low wakeup latencies [28]. Similarly, Blink advocates the use of server low power states that essentially turn off servers[41]. However, OLDI workloads are not amenable to having individual nodes turned off. Meisner et. al showed that coordinated full-system low power modes are needed to achieve energy proportionality for OLDI workloads [29], which is the approach we take. Managing MapReduce clusters for energy efficiency has also been widely explored[24, 22, 9], but these consolidation-based techniques are not applicable for OLDI workloads with tight latency constraints and large dataset footprints.

There are many studies on optimizing energy efficiency of CPUs through the use of DVFS. However, many of those approaches are not applicable to OLDI workloads. Many proposals are geared towards batch workloads without latency requirements. Namely, these DVFS systems improve the energy efficiency of throughput oriented jobs, but they do not make any guarantees on the absolute latency of the job nor do they use the observed latency to inform the DVFS decision [18, 45, 35, 43, 21, 23, 16]. This is not acceptable for OLDI workloads, where the SLO latency is critical. There are also DVFS schemes that aim to maximize energy efficiency of real-time systems where jobs have deadlines [30, 6, 42, 26, 38]. However, these systems have several limitations that make them impractical for OLDI workloads. The first is that both the number and length of jobs must be known ahead of time, which conflicts with the dynamic nature of OLDI workloads. The second is that these control schemes are designed for single node systems; this is not a good fit for OLDI workloads, which run on potentially thousands of servers.

Another way to improve energy efficiency is to use otherwise idle resources on OLDI servers to perform useful work. For example, if *search* operates at 50% utilization, we can fill up the remainder of the cluster with batch workloads that would otherwise be running on other servers. This approach improves energy efficiency by operating servers in their most power efficient mode and can save on capital expenses for batch servers. However, workload co-location has several challenges, including resource provisioning (e.g., removing memory from the search index so that batch workloads can use it) and interference management so that SLO violations are avoided [32, 19, 39, 10, 14, 27, 46]. Investigation of the interactions of co-location with *iso-latency* is beyond the scope of this work.

7 Conclusions

We presented PEGASUS, a feedback-based controller that implements *iso-latency* power management policy for large-scale, latency-critical workloads: it adjusts the power-performance settings of servers in a fine-grain manner so that the overall workload barely meets its latency constraints for user queries at any load. We demonstrated PEGASUS on a Google search cluster. We showed that it preserves SLO latency guarantees and can achieve significant power savings during periods of low or medium utilization (20% to 40% savings). We also es-

tablished that overall workload latency is a better control signal for power management compared to CPU utilization. Overall, *iso-latency* provides a significant step forward towards the goal of energy proportionality for one of the challenging classes of large-scale, low-latency workloads.

8 Acknowledgements

We sincerely thank Alex Kritikos, Chris Johnson, Mahesh Palekar, Nadav Eiron, Parthasarathy Ranganathan, and Peter Dahl for their help and insight for making our work possible at Google. We also thank Caroline Suen and the anonymous reviewers for their useful feedback on earlier versions of this manuscript. This work was supported by a Google research grant, the Stanford Experimental Datacenter Lab, and the Stanford Pervasive Parallelism Lab.

References

[1] [Online]. Available: http://lxr.free-electrons.com/source/drivers/idle/intel_idle.c

[2] Intel®xeon®processor e5-2667 v2 (25m cache, 3.30ghz). [Online]. Available: http://ark.intel.com/products/75273/Intel-Xeon-Processor-E5-2667-v2-25M-Cache-3_30-GHz

[3] "memcached," http://memcached.org/.

[4] "Intel®core™i7 processor family for the lga-2011 socket," vol. 1, 2012.

[5] "Intel®64 and ia-32 architectures software developer's manual," vol. 3B: System Programming Guide, Part 2, Sep 2013.

[6] Hakan Aydin *et al.*, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, 2004.

[7] Luiz Barroso *et al.*, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, Dec. 2007.

[8] Luiz André Barroso *et al.*, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed., 2013.

[9] Yanpei Chen *et al.*, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Proc. of the 7th ACM european Conf. on Computer Systems*, 2012.

[10] Henry Cook *et al.*, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *Proc. of the 40th Annual Intl. Symp. on Computer Architecture*, ser. ISCA '13, 2013.

[11] Intel Corporation, "Addressing power and thermal challenges in the datacenter."

[12] Jeffrey Dean, "Challenges in building large-scale information retrieval systems: Invited talk," in *Proc. of the 2nd ACM Intl. Conf. on Web Search and Data Mining*, ser. WSDM '09, 2009.

[13] Jeffrey Dean *et al.*, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, Feb. 2013.

[14] Christina Delimitrou *et al.*, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," in *Proc. of the 18th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, USA, 2013.

[15] Krisztián Flautner *et al.*, "Vertigo: Automatic performance-setting for linux," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, Dec. 2002.

[16] Rong Ge *et al.*, "Cpu miser: A performance-directed, run-time system for power-aware clusters," in *Parallel Processing, 2007. ICPP 2007. Intl. Conf. on*, 2007.

[17] Michael Huang *et al.*, "A framework for dynamic energy efficiency and temperature management," in *Proc. of the 33rd Annual ACM/IEEE Intl. Symp. on Microarchitecture*, ser. MICRO 33, 2000.

[18] Canturk Isci *et al.*, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," *2012 45th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, vol. 0, 2006.

[19] Ravi Iyer *et al.*, "Qos policies and architecture for cache/memory in cmp platforms," in *Proc. of the 2007 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07, 2007.

[20] N.E. Jerger *et al.*, "An evaluation of server consolidation workloads for multi-core designs," in *Workload Characterization, 2007. IISWC 2007. IEEE 10th Intl. Symp. on*, 2007.

[21] Masaaki Kondo *et al.*, "Improving fairness, throughput and energy-efficiency on a chip multiprocessor through dvfs," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, Mar. 2007.

[22] Willis Lang *et al.*, "Energy management for mapreduce clusters," *Proc. of the VLDB Endowment*, vol. 3, no. 1-2, 2010.

[23] Jungseob Lee *et al.*, "Optimizing throughput of power- and thermal-constrained multicore processors using dvfs and per-core power-gating," in *Proc. of the 46th Annual Design Automation Conf.*, ser. DAC '09, 2009.

[24] Jacob Leverich *et al.*, "On the energy (in)efficiency of hadoop clusters," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, Mar. 2010.

[25] Jacob Leverich *et al.*, "Reconciling High Server Utilization and Sub-millisecond Quality-of-Service," in *SIGOPS European Conf. on Computer Systems (EuroSys)*, 2014.

[26] Caixue Lin *et al.*, "Improving soft real-time performance through better slack reclaiming," in *Proc. of the 26th IEEE Intl. Real-Time Systems Symp.*, ser. RTSS '05, 2005.

[27] Jason Mars *et al.*, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proc. of the 44th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, ser. MICRO-44 '11, 2011.

[28] David Meisner *et al.*, "Powernap: Eliminating server idle power," in *Proc. of the 14th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV, 2009.

[29] David Meisner *et al.*, "Power management of online data-intensive services," in *Proc. of the 38th ACM Intl. Symp. on Computer Architecture*, 2011.

[30] Padmanabhan Pillai *et al.*, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. of the Eighteenth ACM Symp. on Operating Systems Principles*, ser. SOSP '01, 2001.

[31] Johan Pouwelse *et al.*, "Dynamic voltage scaling on a low-power microprocessor," in *Proc. of the 7th Annual Intl. Conf. on Mobile Computing and Networking*, ser. MobiCom '01, 2001.

[32] Moinuddin K Qureshi *et al.*, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. of the 39th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, 2006.

[33] Arun Raghavan *et al.*, "Computational sprinting," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th Intl. Symp. on*, 2012.

[34] Ramya Raghavendra *et al.*, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proc. of the 13th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII, 2008.

[35] K. Rajamani *et al.*, "Power-performance management on an ibm power7 server," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE Intl. Symp. on*, 2010.

[36] Parthasarathy Ranganathan *et al.*, "Ensemble-level power management for dense blade servers," in *Proc. of the 33rd Annual Intl. Symp. on Computer Architecture*, ser. ISCA '06, 2006.

[37] Efi Rotem *et al.*, "Power management architecture of the 2nd generation intel®core™microarchitecture, formerly codenamed sandy bridge," ser. Hot Chips 2011, 2011.

[38] Sonal Saha *et al.*, "An experimental evaluation of real-time dvfs scheduling algorithms," in *Proc. of the 5th Annual Intl. Systems and Storage Conf.*, ser. SYSTOR '12, 2012.

[39] Daniel Sanchez *et al.*, "Scalable and efficient fine-grained cache partitioning with vantage," *IEEE Micro*, vol. 32, no. 3, 2012.

[40] Eric Schurman *et al.*, "The user and business impact of server delays, additional bytes, and http chunking in web search," *Velocity*, 2009.

[41] Navin Sharma *et al.*, "Blink: Managing server clusters on intermittent power," in *Proc. of the 16th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI, 2011.

[42] Youngsoo Shin *et al.*, "Power conscious fixed priority scheduling for hard real-time systems," in *Design Automation Conf.*, 1999.

[43] V. Spiliopoulos *et al.*, "Green governors: A framework for continuously adaptive dvfs," in *Green Computing Conf. and Workshops (IGCC), 2011 Intl.*, 2011.

[44] Niraj Tolia *et al.*, "Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble." *HotPower*, vol. 8, 2008.

[45] Q. Wu *et al.*, "Dynamic-compiler-driven control for microprocessor energy and performance," *Micro, IEEE*, vol. 26, no. 1, 2006.

[46] Hailong Yang *et al.*, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *Proc. of the 40th Annual Intl. Symp. on Computer Architecture*, ser. ISCA '13, 2013.