NapSAC: Design and Implementation of a Power-Proportional Web Cluster

Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, Randy Katz
Computer Science Division
University of California, Berkeley
{krioukov, prmohan, alspaugh, laurak, culler, randy}@cs.berkeley.edu

ABSTRACT

Energy consumption is a major and costly problem in data centers. A large fraction of this energy goes to powering idle machines that are not doing any useful work. We identify two causes of this inefficiency: low server utilization and a lack of power-proportionality. To address this problem we present a design for an power-proportional cluster consisting of a power-aware cluster manager and a set of heterogeneous machines. Our design makes use of currently available energy-efficient hardware, mechanisms for transitioning in and out of low-power sleep states, and dynamic provisioning and scheduling to continually adjust to workload and minimize power consumption. With our design we are able to reduce energy consumption while maintaining acceptable response times for a web service workload based on Wikipedia. With our dynamic provisioning algorithms we demonstrate via simulation a 63% savings in power usage in a typically provisioned datacenter where all machines are left on and awake at all times. Our results show that we are able to achieve close to 90% of the savings a theoretically optimal provisioning scheme would achieve. We have also built a prototype cluster which runs Wikipedia to demonstrate the use of our design in a real environment.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Design, Experimentation

Keywords

Energy, Cluster, Web Server, Web Application, Power Proportional, Power Management, Data Center, Heterogenous Hardware

Green Networking 2010, August 30, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0196-1/10/08 ...$10.00.

1. INTRODUCTION

With the rise of giant web services such as Google, Amazon, Facebook and Wikipedia, energy consumption in data centers has skyrocketed. The EPA estimates that U.S. data centers consumed 61 billion kilowatt-hours in 2006 at a cost of about $4.5 billion. Under current trends this number is expected to double by 2011 [22]. Moreover, the cost of powering a server is approaching the cost of the server hardware itself [2]. Current solutions focus on reducing data center PUE, for example, by reducing the power used for cooling [19] [20]. However, it is how to reduce the base—the energy used by the servers—that is the fundamental question.

The energy consumption of servers is a product of the amount of work and the efficiency with which it is performed. For web service workloads, the amount of work is primarily determined by the rate of user requests, which can vary drastically from peak to average. In a one-week request trace that we obtained from Wikipedia, we observed a peak that was 1.6 times the average rate, though peaks as large as 19 times the average have also been observed [21]. Web service operators like Wikipedia must at the very least provision for the observed peak, taking "...the busiest minute of the busiest hour of the busiest day and build[ing] capacity on that [16]". However, most clusters provision for far more than the observed peak in order to provide a safety margin against flash crowds. As a rough rule of thumb for comparisons in this paper, we conservatively assume that the service is provisioned for twice the observed one-week peak.

A consequence of the gap between peak and average requests rates in workloads, amplified by overprovisioning, is that much of the time many servers sit at very low levels of utilization. In a study of over 5,000 of its servers, Google found that the average CPU utilization for most servers is between 10% and 50% of maximum utilization [3]. At low levels of utilization a server is highly energy-inefficient because the power consumed, even when idle, is over 50% of its peak power even for specially engineered platforms and often over 80% for commodity products [10]. Thus, these servers are not power-proportional, as the amount of power used is proportional to the provisioned capacity, not the request rate. To make this service power-proportional, we must harness the idleness for energy savings.

This problem can be tackled in two ways: by improving the energy efficiency of the entire server hardware platform, or by managing server idleness in software. One hardware technique supported by many CPUs is Dynamic Voltage/Frequency Scaling (DVFS). However, overall power savings achievable with DVFS are small because the power used by the CPU is just a portion of overall server power usage [3]. In addition, if we want to solve this problem using today's hardware, we must use a software solution. Thus, this is the approach we take.

We present our power-aware cluster manager - NapSAC which performs Server Actuation and Control on a heterogeneous set of machines. Our design makes use of currently available energy-efficient hardware, low-power sleep states and the capability to drop into standby and Wake-On-LAN, along with dynamic provisioning and scheduling to aggregate idleness into coarse-grain blocks. With our design we are able to reduce energy consumption while maintaining acceptable response times for a web service workload based on Wikipedia. We provide simulation results that show that under the 2x provision rule we are able to achieve 90% of the savings that a theoretical optimal scheme would achieve by using the bare minimum number of machines needed to serve the load. Even when compared to a system provisioned only for the peak observed load, we are still able to achieve 63% of what the optimal would achieve, while minimally impacting performance. In addition, we have built a prototype cluster which runs Wikipedia to demonstrate the use of our design in a real environment.

2. RELATED WORK

Many CPUs have support for Dynamic Voltage/Frequency Scaling (DVFS) which can be used to dynamically reduce CPU performance and save energy when load is low. Many papers have explored policies for reducing CPU power with DVFS [23, 12]. In [15], Lorch et al. use predictive models to estimate future load and create a power schedule. Unfortunately, the CPU currently contributes less than 50% to overall system power [3], thus we focus on whole system power management.

Virtual machines can be used to dynamically add or remove machines in response to change in load. Several recent papers have used machine learning to dynamically provision virtual machines while maintaining quality of service goals [4, 14]. Virtual machines take minutes to boot or migrate and introduce performance overheads. In contrast to this work we operate at a granularity of seconds, giving us more agility in dealing with sharp changes in request rate.

Several papers have proposed putting machines to sleep. PowerNap [17] models how quickly servers would have to transition in and out of sleep states to achieve energy proportionality on web and other workloads. The authors find that a transition time of under 10ms is required for significant power savings, unfortunately, sleep times on current servers are two orders of magnitude larger. In contrast, we build a power-proportional cluster with current hardware by using predictive provisioning. Chase et. al [6] propose an economic model for allocating resources in a hosting center and putting unused servers into sleep states. A key difference is that our provisioning algorithm explicitly optimizes for energy efficiency utilizing a heterogeneous cluster. We find this heterogeneity essential for obtaining large energy savings with acceptable performance. Gong et. al [7] also trade off user experience with energy usage in the data center. Their work specifically deals with long lived connections as in instant messengers while we concentrate on short lived request-response type of workloads. Pinheiro et. al [18] apply dynamic provisioning strategies to web clusters. However the use of a heterogeneous set of server types and power states that have fast transition times allow us to handle load spikes much better. Finally, we recreate a cluster against a real production class application (Wikipedia) and also evaluate the tradeoffs between user experience and energy usage on traces captured on the Wikipedia servers.

Recent work by Andersen et. al [1] has proposed using embedded processors for reducing energy consumption in I/O based workloads. We also believe that embedded and mobile processors have a place in the data center, however, we consider a broader class of workloads which cannot be run exclusively on embedded systems. Chun et. al [8] make the case for hybrid data centers with the observation that some applications have significantly different performance per watt on different platforms. We agree with this vision and we measure a substantial energy savings when using a heterogeneous cluster over a homogeneous one.

3. DESIGN

3.1 Hardware

An important part of constructing an energy efficient cluster is choosing the proper building blocks. Unfortunately, current systems are not efficient for a wide range of utilization due to high idle power, hence, to attain efficiency at all levels of utilization we must construct a heterogeneous cluster.

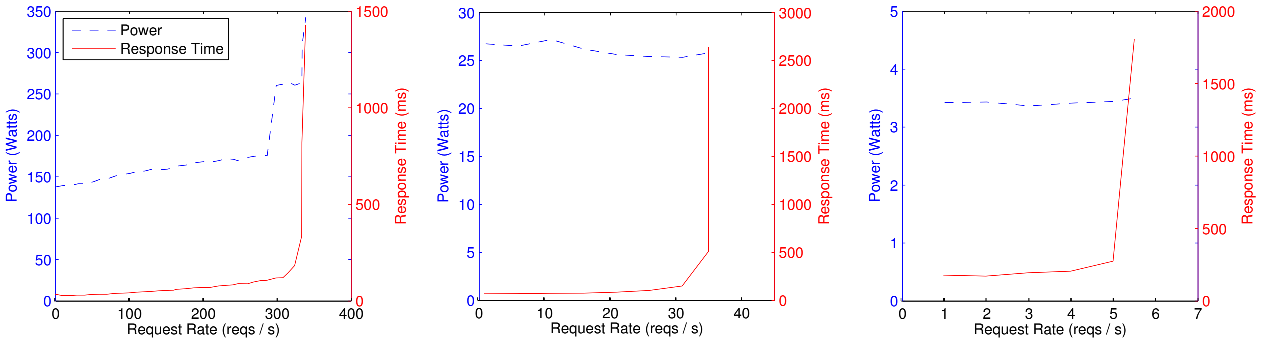
We have two main objectives in selecting hardware. First, we want efficiency at high utilization, and second, we require low-power sleep states with fast transition times to and from these states. The Advanced Configuration and Power Interface (ACPI) specification defines two types of low-power states: idle states (C states) and sleep states (S states) [13]. Idle states reduce solely processor power consumption by disabling the CPU clock, cache and other on-chip circuitry. They are characterized by fast transition times and are generally handled automatically by Linux. The sleep states, on the other hand, include such states as suspend to RAM and hibernate. In these states most of the system is powered down except for the network card which remains active to support Wake-on-LAN (WOL). The system state is maintained either by keeping RAM active or by copying the contents of RAM to persistent storage.

We explore three classes of machines: server, mobile and embedded. To evaluate these machines we use a set of microbenchmarks. Specifically, we run the MediaWiki application serving a copy of Wikipedia. We measure the response times and power consumption of each system at varying request rates. Table 1 shows the systems we evaluated and a summary of their characteristics. Figure 1 shows the response times and power usage of our three platforms running MediaWiki. We also show the efficiency of each system, which we compute by dividing the request rate by power to yield requests per Joule, in Figure 2.

In the server class, we choose to evaluate a high-end machine with two Intel Xeon X5550, Nehalem architecture, quad-core CPUs. This server, like nearly all commercially available servers, has high power consumption and no support for suspend to RAM, though it is efficient at high utilization. At peak, it is able to sustain approximately 340 requests per second with a response time of under 500ms and an efficiency of 1.67 requests per Joule. The mobile class platform we use, based on the Intel Atom 330 CPU,

	Specifications	Low Power States	Peak Power	Idle Power	Max Req Rate	Peak Efficiency
Nehalem Server	2x Intel Xeon X5550 Quad Core	Idle States, Off	248 W	149 W	340 req/s	1.67 req/J
Atom Mobile	Intel Atom 330 Dual Core CPU	Idle States, Off, Suspend to RAM	28 W	22 W	35 req/s	1.25 req/J
BeagleBoard Embedded	TI OMAP3530 ARM Based CPU	Off	3.4 W	3.4 W	5 req/s	1.47 req/J

Table 1: Summary of systems from different classes and their power and performance characteristics.



(a) Nehalem Server

(b) Atom Mobile

(c) BeagleBoard Embedded

Figure 1: Power consumption and response times of the three systems as the request rate increases.

does support suspend to RAM and can be brought in and out of sleep in 2.4 seconds. This system does not have the highest efficiency at peak, but its low-power states and transition times make it agile and efficient at lower utilization. Finally, our ARM processor-based embedded system, the BeagleBoard[9] is very low powered, consuming only 3.4 Watts, but it does not have sufficient performance to run MediaWiki at reasonable request rates. However, the BeagleBoard is useful for serving static content, especially while larger machines are brought up from sleep.

3.2 Architecture

Our system is designed for web service applications in a standard three-tier architecture [5]. Figure 3 shows the system architecture. At the front-end, a set of machines run a high performance load balancer to distribute load among a set of application servers. The application servers are stateless and generate the requested content using data from the storage layer. This architecture is generic and can support different applications and storage systems. It also supports caching at any level in the system.

The key addition of our system is the cluster manager component which provisions for the current load while minimizing power consumption by dictating the sleep state of each machine in the cluster. It interacts with the load balancer to monitor incoming request rates and redirect work to machines in the AWAKE state. NapSAC will put machines to sleep when the incoming request rate is low. A machine is transitioned to the TO-SLEEP state, the load balancer is updated so that no new requests are forwarded to that machine, current requests are allowed to drain, and finally the machine is put into suspend to RAM and marked as being in the SLEEPING state.

For the load balancer we use the low-level software-based Linux Virtual Sever (LVS) [24]. LVS supports a direct routing mode where front-end machines forward packets at the Ethernet layer. All application servers and the load balancer share an IP address. However, only the load balancer responds to ARP requests for the IP address, ensuring that all traffic destined for the system will pass through it. When a new request arrives the load balancer chooses an application server and forwards the request in an Ethernet frame addressed to the chosen machine. Finally, the response is sent directly from the application server to the requester. This design supports a large number of connections and is oblivious to packet contents. Persistent HTTP connections [11] are also supported.

3.3 Dynamic Provisioning

NapSAC dynamically provisions for the current workload by determining the set of servers that should be awake at any point in time. This is calculated with an approximate measure of the number of requests in the current workload that can be handled by each server while still maintaining acceptable response time levels. We assume that the proportions of static and dynamic requests in the workload are more or less constant over time. The servers provide dynamic feedback to the manager about the resource usage on each of the nodes. For our current implementation, we have restricted ourselves to associating a fixed value to the number of requests that can be handled by each server. This value was determined experimentally from the knee of the throughput and response time curves in Figure 1. We at-

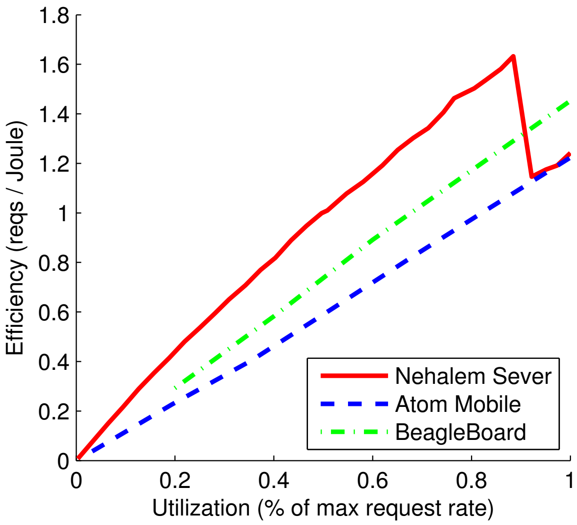


Figure 2: Efficiency of the three systems at varying request rates. Obtained by dividing the request rate by power.

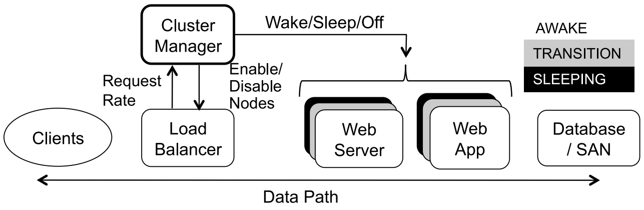


Figure 3: System architecture

tempt to operate each server near but strictly below this maximum request rate. In simulation, we label any requests sent at a higher rate than the knee as "violations" because on a real system they may have response times that violate the desired maximum latency.

The cluster manager runs a greedy *KnapSack* algorithm in order to provision a near optimal number of servers at any given point in time. The optimal provisioning allows each server to operate near but below the maximum request rate with enough slack to absorb unexpected bursts of traffic. This yields power proportionality while ensuring that response time is not impacted even if bursts occur. The online algorithm uses a configuration parameter to determine how much slack to keep for absorbing bursts. In simulation, we compare this to an oracle which, knowing the future workload, can keep the minimum slack necessary for handling all requests.

The algorithm listed in Listing 1 is run every second to determine the set of nodes that should be transitioned to the AWAKE state. Our set of heterogeneous machines is sorted by platform type in order of decreasing performance per watt at peak utilization. The algorithm tries to use the larger server nodes when possible because of their higher performance, and use the mobile and embedded platforms to handle spikes because of their agility. The shut-down algorithm in Listing 2 also kicks in every second and transitions nodes which are no longer needed into the SLEEPING state, starting with those nodes that have the least efficiency.

Listing 1: Server startup algorithm

```
for server_type in server_types:
  start_time = server_type.startup_time
  # Predict and start servers
  pred = predict_load (now + start_time)
  clust = make_cluster(pred)
          - current_cluster
          - nodes waking up in time
  start server_type servers in clust
```

Listing 2: Server turn off algorithm

```
for server_type in rev(server_types):
  start_time = server_type.startup_time
  max_clust = empty_cluster
  for t in range(1, start_time):
      pred = predict_load(now + t)
      temp_clust = make_cluster(pred)
      max_clust = max(max_clust,
                temp_clust)
  temp_clust = current_cluster - max_clust
  turn off servers in temp_clust
```

The transition to and from sleep states is not instantaneous, so the manager predicts the request rate into the future in order to have enough machines in the AWAKE state to handle traffic spikes. The Atom nodes take 2.4 seconds to move from the SLEEP to the AWAKE state. Since the Nehalem server does not support the sleep states we require, we opt instead to power it down in place of putting it to sleep. It takes about 60 seconds to power up. We evaluate the following algorithms for predicting the request rate. A discussion of the evaluation follows in Section 4.1.

Last Arrival (LA) predicts all future values to be the exact same request rate as was last seen.

Moving Window Average (MWA) maintains the request rates seen over the last 'n' seconds and averages the values in the window for the prediction.

Exponentially Weighted Average (EWA) computes the exponentially weighted average using an exponentiation factor α:

$$\texttt{pred} = \alpha\ \texttt{last-req-rate} + (1 - \alpha)\ \texttt{prev-EWA-val}$$

EWA Change Rate (EWA-CR) performs a weighted average on the rate of change of request rates and predicts that the rate of change will be maintained for the prediction time.

3.4 Scheduler

For the current web service architecture we are using, the role of the scheduler is straightforward once we have a well-provisioned cluster. In more complex architectures, for instance, those with distributed state or different types of application servers, and with more complex workloads, for instance, those with batch processing jobs, scheduling becomes a much more challenging problem. In the current system, however, the load balancer uses a weighted round robin approach to distribute load, with the larger machines weighted proportionally more than the less powerful nodes. This policy allows us to maintain the utilization of those machines that are awake to be approximately the same, thus maintaining similar response times.

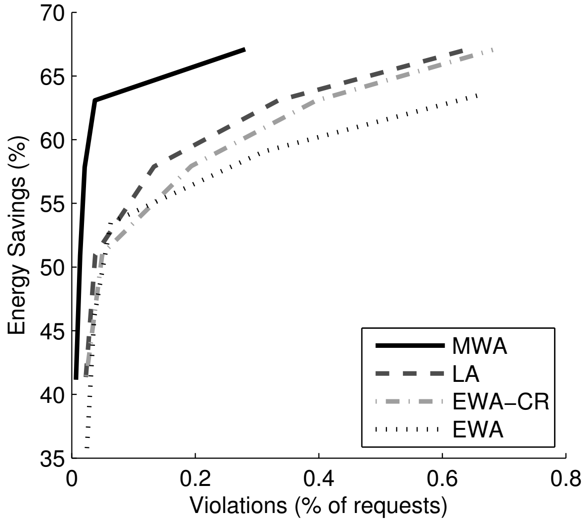


Figure 4: Comparison of different request rate prediction algorithms

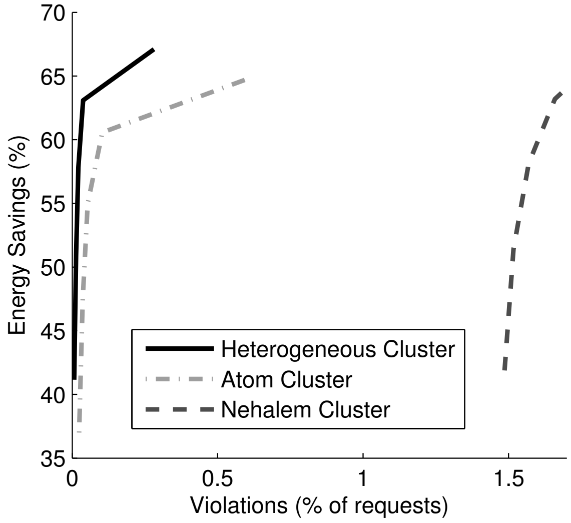


Figure 5: Comparison of MWA on heterogeneous and homogeneous clusters

4. EVALUATION

4.1 Results

We use 7 days of HTTP traffic data from a trace consisting of a 10% sample of the total Wikipedia.org traffic[21] as our workload for comparison. We run the trace in a simulation program, used to compare and evaluate provisioning policies. For simulation, we assume a fixed response time and a maximum request rate per system. These parameters are set empirically by using microbenchmarks. We take the maximum request rate with a response time of less than 300ms for the the Wikipedia workload on each system. We also use power estimates based on the results in Figure 1.

We use a simulator to compare the different server provisioning algorithms in Figure 4. The figure shows a tradeoff between energy savings and the number of "violations." As seen in Figure 1, when the load on the cluster increases beyond the knee of the response time curve, the response times increase dramatically, thus we attempt to always operate below this point. We label all requests that come in at a rate above the knee as "violations" to indicate that they may have response times that are above the desired maximum.

The standard approach to capacity planning is to provision for twice the peak load, and the y-axis of Figure 4 denotes the savings over this baseline. We vary how aggressive the cluster manager is in turning off machines along the length of the curve. This yeilds a range of tradeoffs between energy savings and the number of "violations." At the lower-left point of the curve the cluster manager is most conservative with few violations but also lower energy savings; moving to the upper-right, it becomes more aggressive with more violations and improved energy savings. We note that the moving window average performs the best. When compared to an oracle provisioning algorithm, which always achieves maximal utilization by knowing future workload, the moving window average performs within 10% of optimal.

It achieves about 63% energy savings compared to a typical cluster provisioned at twice its peak rate with less than 0.03% of requests potentially slowed down. The energy savings is still 27% when compared to a cluster provisioned exactly for the peak usage.

A key feature of our cluster is the use of a heterogeneous mix of hardware. By using multiple classes of hardware we can use the best features of each system. As was noted earlier, server class systems do not usually support sleep states, requiring length shutdown and boot cycles for energy savings. On the other hand, the mobile class Atoms devices support suspend-to-RAM and have wake up times of about 2.4 seconds. However, at peak utilization server systems are typically more energy efficient than mobile systems. Thus, we choose to use the Atom devices to handle sharp bursts of traffic and the Nehalem servers for the base load. In Figure 5, we compare the efficiency of the MWA algorithm for different types of clusters; homogeneous clusters of only Nehalems or Atoms and a mixture of both. We clearly see that heterogeneous cluster outperforms either homogenous option, allowing higher energy savings with fewer violations. Although a homogeneous cluster of Atom machines performs close to the heterogeneous mixture in terms of energy savings, it is not a valid proposition because of the physical space it uses and its inapplicability in certain classes of applications such as large database servers.

While a custom Atom node design might allow for dense packing of mobile nodes, our Atom node uses an off-the-shelf development board, which still takes up a large amount of space. Thus, we cannot take advantage of the Atoms' lower power to attain the same power density of a server's by putting more nodes into a smaller space. Additionally, more nodes equates to more required network infrastructure, which itself requires more power along with more complicated wiring and management. We introduced BeagleBoards and looked at their efficiency and performance but do not utilize them in these experiments because they suffer the same density problems as the Atom cluster.

4.2 Experiment Setup

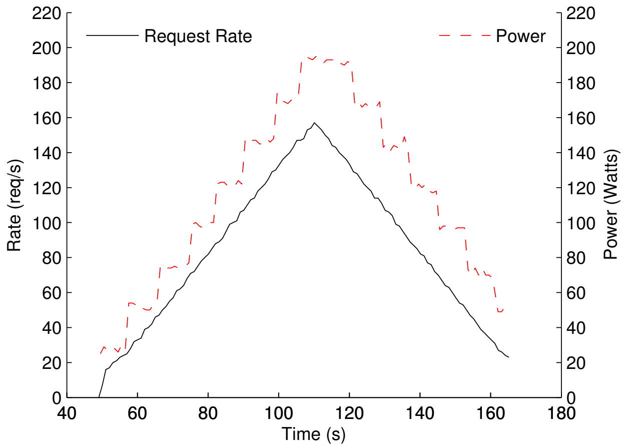


Figure 6: Power-proportionality for a simple traffic spike.

We use a real application, Wikipedia, to evaluate our energy efficient cluster. Wikipedia is based on MediaWiki which has a tiered architecture; application servers running PHP code and a backend database. We choose one Nehalem server machine to host the backend database to be shared by all application servers. This server runs MySQL 5.1.31 loaded with a copy of the Wikipedia article data. Next, we setup 16 Atom machines to act as application servers. Each system runs Linux 2.6.28 with a patch to disable ARP as required for the load balancer and explained in Section 3.2. We install the PHP 5.3.2 interpreter with opcode caching and setup Lighttpd as the web server. Finally, an Intel Core 2 Duo based desktop is used as the load balancer and cluster manager.

To evaluate our cluster manager we generate two traffic spikes that stress the manager's ability to scale up and down in response to load. For each traffic pattern we select files from the Wikipedia trace distribution. The first trace shown in Figure 6 simply ramps the request rate up and down linearly. As expected, the cluster manager wakes up additional systems when the rate increases and turns them back off as the rate drops. 99.9% of the requests complete in 60 milliseconds. Overall, the system uses 29.4% of the energy needed by cluster provisioned at twice its peak rate or 58.9% of the energy used by a cluster provisioned exactly for peak.

For the second experiment we choose a more complex workload consisting of a sinusoidal pattern with sharp noise that is more representative of web traffic. Figure 7 shows the power used by the cluster executing this workload. The cluster manager is able to smoothly ramp up and down the number of awake machines despite the noisy traffic pattern. This is largely due to the smoothing effect of the rate predictor. 99.9% of the requests complete within 78 milliseconds and all requests complete within 200 milliseconds. Overall, the system uses 31.8% of the energy needed by cluster provisioned at twice its peak rate.

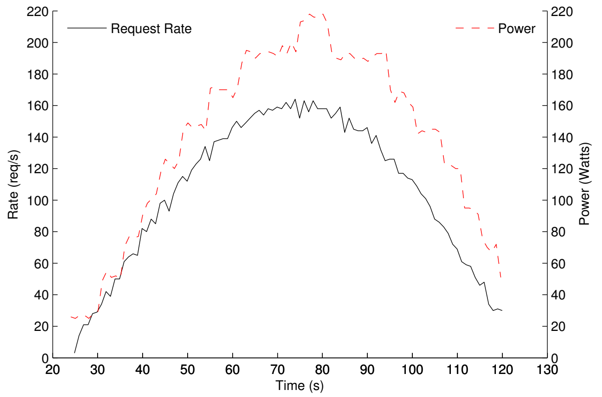


Figure 7: Power-proportionality for a noisy sinusoidal traffic pattern.

5. CONCLUSION

We have presented a design for a power-proportional cluster using a power-aware cluster manager and a set of heterogeneous machines that harnesses idleness to achieve significant energy savings. We have built a cluster as a three-tier web service out of three different types of hardware, representing server-, mobile-, and embedded-class processors to demonstrate the use of our design in a real environment.

We have compared via simulation several different provisioning algorithms for determining the necessary set of machines to handle a Wikipedia-based workload. The greatest power savings came from using a Moving Window Average scheme, which resulted in a 27% savings over the power usage of a cluster provisioned to exactly meet it's peak demand. Note that even greater savings will be achieved in workloads with higher peak-to-average request rate ratios. Our scheduling algorithms are relatively simple yet still manage to save a significant amount of energy. In the future, we plan on looking at more sophisticated predictors which will allow the scheduler to handle a greater variety of workloads. Our results show that under the 2x provision rule we are able to achieve 90% of the savings that a theoretical optimal scheme using an oracle scheduler with future knowledge would achieve. In addition, these savings can be achieved while minimally impacting performance. As a large proportion of servers in the real world are idle for large portions of the day, we expect our system design to enable major energy savings when deployed at large scale.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. We also thank Albert Goto, Jeff Anderson Lee and Ken Lutz for helping us set up the cluster and solve various deployment hurdles. This work was supported in part by NSF Grant #CPS-0932209, the FCRP MuSyC Center, and Intel and Samsung Corporations. The opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporting institutions.

6. REFERENCES

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: A fast array of wimpy nodes. *SOSP '09*, October 2009.

[2] L. A. Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

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

[4] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. *HotCloud'09*, 2009.

[5] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centres. In *SOSP*, pages 103–116, 2001.

[7] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. 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*, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[8] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *HotPower'09*, 2009.

[9] G. Coley. Beagleboard system reference manual, October 2009.

[10] S. Dawson-Haggerty, A. Krioukov, and D. E. Culler. Power optimization - a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct 2009.

[11] Fielding, et al. Hypertext transfer protocol - http/1.1. *Network Working Group*, 1999.

[12] D. Grunwald, P. Levis, K. I. Farkas, C. B. M. III, and M. Neufeld. Policies for dynamic clock scheduling. In *OSDI*, pages 73–86, 2000.

[13] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification, June 2009.

[14] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC '08:*, 2008.

[15] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In *SIGMETRICS/Performance*, pages 50–61, 2001.

[16] D. McCullagh. Turbotax e-filing woes draw customer ire. 2007.

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

[18] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. pages 75–93, 2003.

[19] N. Rasmussen. Electrical efficiency modeling of data centers. Technical Report White Paper #113, APC, 2006.

[20] R. K. Sharma, C. E. Bash, C. D. Patel, R. J. Friedrich, and J. S. Chase. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1):42–49, 2005.

[21] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.

[22] U.S. Environmental Protection Agency. Epa report on server and data center energy efficiency. *ENERGY STAR Program*, 2007.

[23] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *OSDI*, pages 13–23, 1994.

[24] W. Zhang. Linux virtual server for scalable network services. *Ottawa Linux Symposium*, 2000.