Barely Alive Memory Servers: Keeping Data Active in a Low-Power State

VLASIA ANAGNOSTOPOULOU, SUSMIT BISWAS, HEBA SAADELDEEN, and ALAN SAVAGE, University of California at Santa Barbara
RICARDO BIANCHINI, Rutgers University
TAO YANG, DIANA FRANKLIN, and FREDERIC T. CHONG, University of California at Santa Barbara

Current resource provisioning schemes in Internet services leave servers less than 50% utilized almost all the time. At this level of utilization, the servers' energy efficiency is substantially lower than at peak utilization. A solution to this problem could be dynamically consolidating workloads into fewer servers and turning others off. However, services typically resist doing so, because of high response times during reactivation in handling traffic spikes. Moreover, services often want the memory and/or storage of all servers to be readily available at all times.

In this article, we propose a family of *barely alive* active low-power server states that facilitates both fast reactivation and access to memory while in a low-power state. We compare these states to previously proposed active and idle states. In particular, we investigate the impact of load bursts in each energy-saving scheme. We also evaluate the additional benefits of memory access under low-power states with a study of a search service using a cooperative main-memory cache. Finally, we propose a system that combines a barely-alive state with the off state. We find that the barely alive states can reduce service energy consumption by up to 38%, compared to an energy-oblivious system. We also find that these energy savings are consistent across a large parameter space.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: System Architectures

General Terms: Design, Management, Performance

Additional Key Words and Phrases: Energy management, active low-power states, memory management

ACM Reference Format:
Anagnostopoulou, V., Biswas, S., Saadeldeen, H., Savage, A., Bianchini, R., Yang, T., Franklin, D., and Chong, F. T. 2012. Barely alive memory servers: Keeping data active in a low-power state. ACM J. Emerg. Technol. Comput. Syst. 8, 4, Article 31 (October 2012), 20 pages.
DOI = 10.1145/2367736.2367742 http://doi.acm.org/10.1145/2367736.2367742

1. INTRODUCTION

Energy represents a large fraction of the operational cost of Internet services. As a result, previous works have proposed approaches for conserving energy in these

This material is based on work supported by a Google Directed Research Grant on Energy-Proportional Computing, NSF-CCF-1017578, and an NSF CAREER award to D. Franklin. T. Yang is supported in part by NSF-IIS-1118106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.
Authors' addresses: V. Anagnostopoulou (corresponding author), S. Biswas, H. Saadeldeen, and A. Savage, Department of Computer Science, University of California at Santa Barbara; email: vlasia@cs.ucsb.edu; R. Bianchini, Department of Computer Science, Rutgers University; T. Yang, D. Franklin, and F. T. Chong, Department of Computer Science, University of California at Santa Barbara.

DOI 10.1145/2367736.2367742 http://doi.acm.org/10.1145/2367736.2367742

services, such as consolidating workloads into a subset of servers and turning others off [Chase et al. 2001; Chen et al. 2005, 2008; Pinheiro et al. 2001], and leveraging dynamic voltage and frequency scaling of the CPUs [Chen et al. 2005; Elnozahy et al. 2003; Fan et al. 2007].

Consolidation is particularly attractive for two reasons. First, current resource provisioning schemes leave server utilizations under 50% almost all the time [Fan et al. 2007]. At these utilizations, server energy efficiency is very low [Barroso and Hölzle 2007]. Second, current servers consume a significant amount of energy even when they are completely idle [Barroso and Hölzle 2007]. Despite its benefits, services typically do not use this technique. A major reason is the fear of high response times during reactivation in handling traffic spikes. Another reason is that services often want the memory and/or storage of all servers to be readily available even during periods of light load. For example, interactive services try to maximize the amount of memory available for data caching across the cluster, thereby avoiding disk accesses or content regeneration.

In this article, we propose an approach that does not completely shutdown idle servers, enables fast state transitions, and keeps in-memory application code/data untouched. Specifically, we propose to send servers to a new family of "barely alive" power states, instead of turning them completely off after consolidation. In a barely alive state, a server's memory (and possibility its disks) can still be accessed, even if many of its other components are turned off. Keeping data active and accessible in barely alive states enables software to implement cluster-wide (or "cooperative") main-memory caching, data replication and coherence, or even cluster-wide in-memory data structures, while conserving a significant amount of energy.

Our evaluation starts by comparing barely alive states to conventional consolidation via complete server shutdown, as well as more recent proposals such as PowerNap and Somniloquy. In particular, we evaluate the effect of server restart latency on response time during typical load spikes. Spikes may occur due to a variety of reasons, including external events (e.g., Slashdot effect), the temporary unavailability of a mirror data center, operator mistakes, or software bugs. Under latency constraints, greater restart latency translates to a larger number of extra active servers provisioned to absorb the load. We evaluate the sensitivity of each energy conserving scheme to the duration and magnitude of load spikes, as well as to modifications to data while in energy-conserving server states.

We then present a case study of a server cluster implementing a cooperative cache for the "snippet" generator of a Web search service. Many services today use cooperative caching middlewares (e.g., Memcached is used at Wikipedia, Twitter, and others [Dormando 2011]). Our cooperative caching implementation accommodates barely alive servers and dynamically resizes the cache as a function of workload variations and desired performance. Any memory not used for caching can be used by applications. For this study, we simulate systems based on an efficient barely alive state, on-off, Somniloquy, and low-end servers. We also investigate the trade-off between performance and energy savings under various system parameters. Finally, we introduce and evaluate a "mixed" system, which combines active, barely alive, and off states. From these studies, we find that, at each performance level, the mixed system achieves the highest energy savings. Overall, barely alive states can produce energy savings of up to 38%, compared to a baseline energy-oblivious system. Moreover, we find that barely alive states can conserve significant energy across a large parameter space.

The remainder of the article is organized as follows. Next, we discuss the background and related work. In Section 3, we introduce the barely alive family of power states. We qualitatively compare our family of states to previous schemes in Section 4. Section 5 presents our analysis of provisioning for load spikes. In this section, we also describe our simulation infrastructure and aggregate memory results. In Section 6, we introduce the mixed system, and assess its energy savings at different performance levels as compared to other approaches. Finally, we draw our conclusions in Section 7.

2. BACKGROUND AND RELATED WORK

Many papers have studied dynamic workload consolidation and server turn off [Chase et al. 2001; Chen et al. 2008, 2005; Heath et al. 2005; Pinheiro et al. 2001; Rajamani and Lefurgy 2003]. The idea is to adjust the number of active servers dynamically, based on the load offered to the service. During periods of less-than-peak load, the workload can be concentrated (either through state migration or request distribution) on a subset of the servers and others can be turned off. In this article, we demonstrate how to make this approach to energy conservation more practical through the creation of a family of active low-power server states.

An orthogonal approach to consolidation and turn off is to dynamically scale the voltage/frequency of the processor (DVFS), when the CPU load is low. We focus on consolidation and turn off for two main reasons. First, DVFS currently only applies to the CPU, while other server components also consume significant power. Second, the opportunity to reduce voltage (the main source of CPU energy savings) has and will continue to diminish over time.

Two recent works have proposed low-power server states [Agarwal et al. 2009; Meisner et al. 2009]. Somniloquy augments the network interface to be able to turn most other components off during periods of idleness, while retaining network connectivity. In the low-power state, main memory becomes inaccessible, so accesses can only be performed to the small memory of the network interface. No disk accesses can be affected. Moreover, updates to main memory can only be performed after activation, thereby increasing delay. In contrast, our states allow read and write accesses to the entire main memory and disks. We compare against Somniloquy extensively in Sections 4 and 5.

PowerNap rapidly transitions servers between active and “nap” state, obviating the need for consolidation. In nap state, a server is not operational. PowerNap requires server software to avoid unwanted transitions to active state (e.g., due to clock interrupts). More challengingly, PowerNap requires the server to be completely idle, which is becoming harder as the number of cores per CPU increases (the idle times of all cores must perfectly overlap). We compare against PowerNap extensively in Sections 4 and 5.

Our study focuses on high-performance servers with consolidated workloads requiring significant processing power. Other work has studied data centers comprising lower performance (and power) servers [Andersen et al. 2009]. These servers were not found to be particularly advantageous for Web search in terms of energy (although more advantageous in terms of cost) in Lim et al. [2008]. More recently, Reddi et al. in Reddi et al. [2010] found that these servers perform poorly for a computationally intensive search engine workload. In Section 5.3, we compare our results to those of such servers.

Some states in the barely alive family turn all the CPU cores off but still allow memory accesses through the network interface. Remote Direct Memory Access (RDMA) also allows memory to be accessed without host intervention. However, the previous works in RDMA have focused on using this mechanism to bypass an active CPU in fully operational servers. A few high-end network interface cards, such as InfiniBand [Liu et al. 2003], provide RDMA capabilities. Although we intentionally abstract the mechanisms required by RDMA (e.g., address registration and memory pinning) in this article, we do rely on similar functionality.

Another approach that enables remote memory accesses in a blade chassis is Disaggregated Memory (DM) [Lim et al. 2009]. In DM, a set of memory blades extend the memory of the compute blades. A memory blade can be seen as a server in a barely alive state with all cores and disks turned off. However, our approach is more flexible in that barely alive servers can be activated and recover the full functionality of a server. In addition, most barely alive states require no hardware modification and can use off-the-shelf clustering software. Finally, in our approach, each server includes more local memory, reducing interconnect bandwidth requirements with respect to DM.

3. BARELY ALIVE STATES

We propose a family of barely alive server states. The states differ in terms of exactly what components are turned off to conserve energy. The unifying characteristic of all states in the family is that selected levels of the memory hierarchy (main memory and possibly disks) can be accessed by remote servers, despite the fact that some components are turned off. An Internet service can transition some servers to one of the barely alive states, instead of the off state, after consolidating the workload on another set of servers.

Members of the family. We have identified many barely alive states, called "BA" followed by a member number. The deepest state, `BA1`, turns off all the cores, all the disks, the shared cache, all but one fan, and all but one network interface. The memory controller is kept on (even if the controller is on chip), but the memory devices are sent to the self-refresh mode immediately after any access. Remote memory accesses occur through a very low-power embedded processor built into the network interface. (Some existing network cards include programmable processors in them, e.g., Myricom [2009].) This processor accesses memory by driving the memory controller directly, just as in regular DMA operations involving the network interface. In fact, compared to current server hardware, the only hardware support required by BA1 is a separate power rail for the memory controller and the low-power embedded processor (if it is not already available in the network card).

`BA2` consumes slightly more power than BA1, as it manages the memory using the standard closed-page policy. Under this policy, most power savings (beyond those of BA1) come from transitioning memory ranks that have no open row buffers to the (precharge) powerdown mode. BA2 requires no hardware support beyond that for BA1.

BA1 and BA2 can be used when the memory access traffic on a barely alive server is low enough that a single network interface and embedded processor can manage. Higher load may require additional components to be activated. In state `BA3`, one or more additional network interfaces are activated. To name variations with different numbers of active components, we use a suffix. For example, when two active network interfaces are used, we refer to this state as `BA3-2NI`. Again, BA3 requires no hardware support beyond that for BA1.

If the load on a barely alive server is excessively high for the embedded processors to handle, one or more cores (and possibly fans) must be activated; the embedded processors can be turned off. State `BA4` represents these scenarios. The deepest of the BA4 states is `BA4-1C`, which keeps a single core, fan, and network interface active. The shared cache is active as well. In terms of hardware support, BA4 requires the ability to turn off cores independently. This ability already exists in some modern multicore CPUs. In addition, BA4 could benefit from the ability to activate only part of the shared cache, for example, $1/N$ of it for an N-core CPU. Current processors do not provide this feature.

One or more cores must also be active, when remote disk accesses to barely alive servers are needed. The active core(s) can execute the device driver for the disk(s).

State `BA5` represents these scenarios. The deepest of the BA5 states is `BA5-1C-1D` which keeps a single core, fan, network interface, and disk active. BA5 requires no hardware support beyond that for BA4.

Transition overheads. The transitions to and from a barely alive state are initiated by a CPU core (if at least one is active) or by the embedded processor (if no core is active). Transitions can be between active state and a barely alive state or between two barely alive states. Regardless of the states involved, transitions can be very fast and consume little energy, since the memory contents (including any cached data and the operating system state) are not affected. In fact, updates to the memory contents can occur while the server is in a barely alive state. The discussion that follows quantifies these overheads for two extreme transitions: (1) from the active state to BA5 and back (disks remain active all the time); and (2) from the active state to BA1 and back (disks can be shutdown in the barely alive state).

The transitions between the active state and BA5 take on the order of microseconds, that is, the time needed to transition the cores and network interfaces. The fans need not complete their transitions before the server can be declared in the barely alive or the active state. The energy overhead of the transitions is negligible.

The transition from active state to BA1 also takes on the order of microseconds, since the fans and disks can complete their transitions in the background. The energy overhead of this transition is dominated by the energy consumed in spinning down the disks. In contrast, the transition from BA1 to the active state is dominated in terms of both time and energy by the disk activation overheads. Carrera et al. [2003] have quantified the overheads of sending an IBM Ultrastar disk to the standby state at 10 Joules and 2 seconds, and the overheads of activating it at 100 Joules and 10 seconds. Others [Weddle et al. 2007] have reported much lower overheads for a Fujitsu disk. Fortunately, these overheads are modest, given that Internet service workloads allow servers to stay in a barely alive state for long periods of time.

Implications for software. To be most useful, the barely alive family requires the cluster software to have the ability to: (1) consolidate the workload into a subset of (active) servers and (2) perform remote memory (read and/or write) accesses to barely alive servers. For Internet services, it would be natural for the cluster software to implement some sort of cooperative main-memory caching system [Carrera and Bianchini 2005; Fitzpatrick 2004; Pai et al. 1998], which would manage the main memories of the cluster as a single large cache. This implementation could be coupled with a standard consolidation algorithm. In fact, regardless of the barely-active state(s) used, the consolidation algorithm can be the same as before [Chase et al. 2001; Pinheiro et al. 2001]. The only adjustment is that schemes involving larger activation overheads (e.g., on-off consolidation) require more servers to be active at all times to handle typical load spikes.

Although cooperative caching is a good application for barely alive servers, other types of data center workloads are also amenable to our family of states. For example, one might implement a distributed file service that sends some servers to a barely alive state under light load, but continues using their memories to avoid disk accesses. Another example is a replicated database system that transitions servers to a barely alive state, but keeps updating the tables they store and/or cache. Even MapReduce computations with limited parallelism can leverage the set of main memories to store large data structures. Obviously, the best barely alive state for these types of workloads may be different than that for cooperative caching.

When using barely alive states in which at least one core is active (e.g., state BA4-1C), all memory addressing can be done using virtual addresses. Furthermore, the disks of barely alive servers can be accessed (e.g., state BA5-1C-1D).

Table I. Comparison of Techniques

System	Access to all memory	Power	Transition time (up/down)	Transition energy (up/down)
Traditional servers	Y	$O(300W)$	N/A	N/A
Low-end servers	Y	$O(50W)$	N/A	N/A
PowerNap	Y	$O(40W)$	$O(\mu s)/O(\mu s)$	$O(\mu J)/O(\mu J)$
BA1	Y	$O(30W)$	$O(10s)/O(\mu s)$	$O(100J)/O(10J)$
BA2	Y	$O(40W)$	$O(10s)/O(\mu s)$	$O(100J)/O(10J)$
BA3-2NI	Y	$O(50W)$	$O(10s)/O(\mu s)$	$O(100J)/O(10J)$
BA4-1C	Y	$O(60W)$	$O(10s)/O(\mu s)$	$O(100J)/O(10J)$
BA5-1C-1D	Y	$O(70W)$	$O(\mu s)/O(\mu s)$	$O(\mu J)/O(\mu J)$
Somniloquy	N	$O(30W)$	$O(10s)/O(\mu s)$	$O(100J)/O(10J)$
On/Off	N	$O(0W)$	$O(100s)/O(\mu s)$	$O(1000J)/O(100J)$

Notes: Transition overheads include the time and energy of transitions, and memory content reloading and updating after activation. Power numbers assume a single CPU and do not include power supply losses. For a fair comparison, we assume that in PowerNap a disk is present but is never shutdown. Table III shows a more detailed breakdown of the power consumptions we assume.

For the family members that turn off all cores (BA1, BA2, and BA3), memory addressing requires careful handling in software. In particular, as the embedded processor does not understand virtual addresses, the remote memory accesses have to specify physical addresses or be translated to physical addresses in software by the embedded processor. Memory management also becomes more difficult when multiple embedded processors are active (e.g., state BA3-2NI). In this case, the software is responsible for guaranteeing proper coordination. Finally, the embedded processor has to implement some sort of (RDMA) communication protocol to be able to receive memory access requests coming from active servers and reply to them. As our target system is a server cluster, this communication protocol can be lean and simple. Because the barely alive states are independent of this protocol, we do not discuss it further.

4. QUALITATIVE EVALUATION OF THE BARELY ALIVE STATES

Table I presents a qualitative comparison of the power consumption and transition overheads to and from active state of the members of the barely alive family. The power numbers assume a single multicore CPU and do not include power supply losses. The table also includes the same characteristics of PowerNap [Meisner et al. 2009], Somniloquy [Agarwal et al. 2009], On/Off [Chase et al. 2001; Pinheiro et al. 2001], and low-end servers (e.g., Atom-based servers) [Andersen et al. 2009; Reddi et al. 2010; Lim et al. 2008]. Table III shows a more detailed breakdown of the power consumptions we assume. In comparing the systems, we assume that they run an Internet service workload and a cluster-wide cooperative caching middleware.

We first describe the systems that rely on load consolidation (the bottom group in the table). The barely alive family was described before. We assume that the content of the memory of a server with no active cores (BA1, BA2, and BA3-2NI in the table) is only updated when the server is activated. Somniloquy is similar to BA1, except that all accesses in the low-power state are performed to memory in the network interface itself, rather than main memory. As a result, data updates are only performed to main memory when the server is activated. In addition, the amount of memory that can be accessed is limited to the size of the network interface memory. These two characteristics mean that Somniloquy must keep more servers active than a barely alive system to compensate for the higher activation time and the smaller global memory cache.

The On/Off system turns servers completely off after consolidation, which means that part of the cluster memory cannot be accessed, server activation takes a long time, and data updates are done in batches after activation. Thus, the On/Off system needs to keep more servers active than a barely alive system to compensate for the smaller memory cache and guarantee that server activation does not translate into higher response times.

PowerNap and low-end servers do not rely on consolidation. PowerNap sends all components (except for disks, which were replaced by solid-state drives in Meisner et al. [2009]) to their deepest power states whenever there is any idle time at a server. Unfortunately, multicore servers are completely idle only for very short periods of time (if at all), since the core idle times have to overlap perfectly. Low-end servers seek to provide better energy efficiency simply through the use of more efficient (and often lower performance) components; no power state changes are effected. As a basis for comparison, we also consider a system that uses traditional 1U servers and keeps them active at all times.

The key observations to make from this table are: (1) all systems have very low-power states with different levels of energy savings; (2) transition overheads are not significant (except in the On/Off system), since we expect the systems that leverage consolidation to transition states at the granularity of hours. Moreover, in the barely alive and Somniloquy systems, the time to activate a server can be reduced from $O(10s)$ to $O(\mu s)$, if the system does not shutdown the disk; (3) when there is idle time at all (i.e., under extremely low utilization), transition frequencies are likely to be high for PowerNap, which would significantly increase the system's energy consumption; and (4) the low power of Somniloquy and On/Off is partially countered by the need to keep more servers active, leading to higher overall energy, as we shall demonstrate in our results.

Overall, the barely alive family presents a range of interesting trade-offs between power and overhead. BA1 is a deep power state with relatively small performance and energy overheads, whereas BA5-1C-1D consumes more power but has trivial overheads. No other system can achieve all the benefits that barely alive states provide.

Although all of the barely alive states support our goals well, henceforth we will focus on BA2 only. This state represents an interesting design point for two reasons: (1) some current processors already have a power rail for the memory controller that is separate from those for the cores; and (2) leaving one core on currently requires leaving the entire shared cache on, reducing energy savings. Both these reasons are illustrated by the Nehalem "uncore" [Shimpi 2008].

5. QUANTITATIVE EVALUATION OF THE BARELY ALIVE STATES

5.1. Benefits of Fast Activation

A significant challenge for all load consolidation schemes is handling typical load spikes without violating latency constraints [Fan et al. 2007; Jung et al. 2009]. In this section, we present a simple analysis of barely alive and previous schemes when faced with a parameterized load spike. We estimate the extra server provisioning and illustrate the trade-offs of activation latency, standby power consumption, and data update latency. We use the intuition deriving from these trade-offs in our detailed case study in Section 5.3.

To avoid excessive latency, extra active servers must be provisioned to absorb the spike load until more servers can be activated. The number of extra active servers must match the typical increase in load during the activation time. In more detail,

$$\begin{aligned} \mathit{NumExtraAct} = {} & (\mathit{MaxLoadRateAfterActTime} - \mathit{LoadRateBeforeSpike})/ \\ & \mathit{ActServerCapacity}, \end{aligned}$$

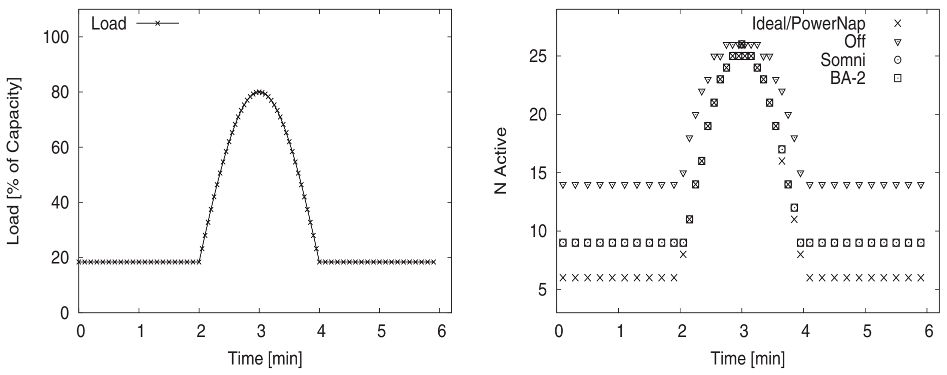

Fig. 1. Load spike (left) and server provisioning (right).

where *NumExtraAct* is the number of extra active servers; *MaxLoadRateAfterActTime* is the maximum request rate during a period equal to the activation time, since the beginning of a typical spike; *LoadRateBeforeSpike* is the request rate before the spike begins; and *ActServerCapacity* is the request processing capacity of an active server. Thus, the higher the latency of server activation, the more the request rate during the spike will increase, and the more extra servers must be provisioned.

Next, we quantify these effects. Our analysis assumes that the cluster has 32 servers, and each active server can process 2000 connections/second and consumes 222–404W as a linear function of utilization. For the latency of server activations, we assume 30s for the On/Off system, and 10s for the BA2 and Somniloquy systems. All these values match our assumptions in the simulation results section.

In Figure 1(left), we present an example of a synthetic load spike, which increases the request load on the system from 20% to 80% of its maximum capacity. Figure 1(right) shows that the number of active servers before the load spike is significantly higher for the On/Off system than for the more sophisticated BA2, PowerNap, and Somniloquy systems. For a baseline comparison, the "ideal" system is an On/Off system in which servers can be brought up with zero latency and no energy overhead. As originally proposed [Meisner et al. 2009], PowerNap exhibits near-zero transition latency. BA2 and Somniloquy are equivalent with respect to load spike provisioning, as long as no data needs to be modified at servers in a low-power state.

We can parameterize load spikes by duration and amplitude, and choose parameters consistent with observed behavior such as from studies of an HP customer's Web server trace [Jung et al. 2009]. Figure 2(left) shows how the power overhead of extra server provisioning (with respect to the ideal system) varies with spike amplitude, assuming a duration of 2 minutes. We can see that the On/Off system entails a modest level of overhead with spikes of low amplitude. However, the overhead grows significantly as the spike increases in amplitude. Figure 2(right) shows the impact of spike duration, assuming an amplitude of 60% of the peak capacity. We can see that, if the duration of the spike is 2 minutes, the overprovisioning overhead is large. The overhead drops to more modest levels for spikes lasting 10 minutes.

5.2. Benefits of Allowing Immediate Data Updates

Services modify data that may reside on servers that are off or in a low-power state. A key advantage of barely alive systems is the ability to directly modify data in main memory while in a low-power state. Modification of such data is impractical in On/Off and PowerNap systems. For On/Off systems, writes would need to be source buffered

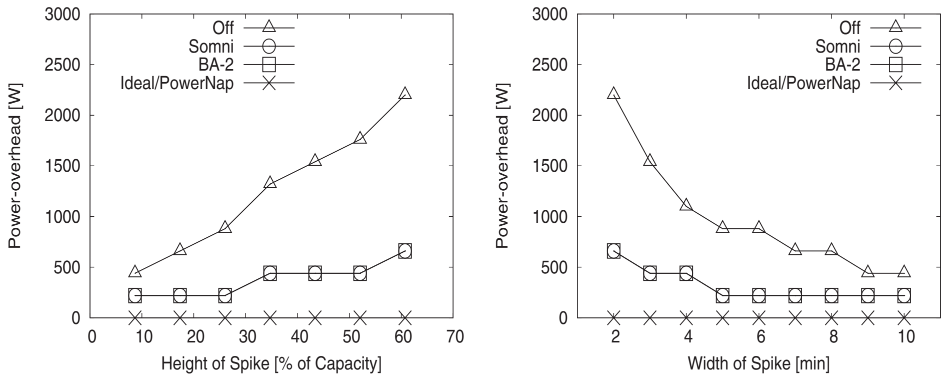

Fig. 2. Impact of spike height (left) and width (right).

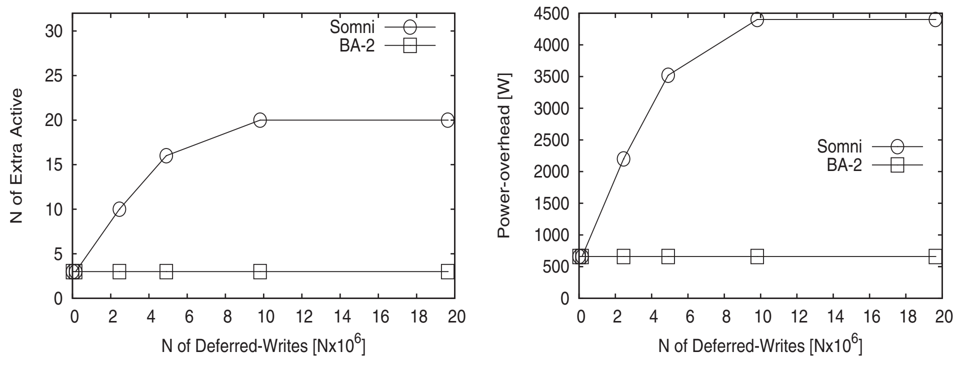

Fig. 3. Impact of deferred writes on the number of extra active servers (left) and power overhead (right).

and deferred to wake up, which can be problematic if systems are off for long periods of time. PowerNap can avoid this problem by waking up the server to perform data updates. However, for all but the most insignificant of write intensities, PowerNap would spend too long in active state.

Other than barely alive, Somniloquy offers the best solution to data updates while in a low-power state. Writes can be buffered in the Somniloquy device. However, with the limited size of the Somniloquy memory (64MB), we assume that writes would need to be buffered in their Secure Digital (SD) card auxiliary storage. The time to read the updated data from the SD card to main memory during activations increases the activation time and, thus, increases the number of extra active servers (Eq. (1)).

In Figure 3, we compare BA2 and Somniloquy as the number of deferred writes varies, assuming the same cluster and server parameters from the previous subsection. Writes are to objects typical of our Web application (6KB each). Figure 3(left) quantifies the number of extra active servers in each system. As the number of buffered writes increases (either due to higher write traffic or longer time in a low-power state), the Somniloquy activation latency becomes significant. This effect quickly results in a large number of extra active servers provisioned for spikes. The number of extra active servers levels out when transitioning servers to Somniloquy state actually starts to increase energy consumption. Figure 3(right) shows the same comparison in terms of total power for extra active servers.

5.3. Benefits of Aggregating Memory

Although barely alive server states provide fast server activation and allow data updates while in a low-power state, an even greater advantage is their ability to effectively use all of a cluster's memory while adjusting processing power to reduced load. In this section, we pair the systems we study with a middleware implementation of distributed cooperative object caching. The middleware manages the available memory resources across the cluster as a single large cache to avoid disk accesses. This set of experiments is motivated by the observation that, while load offered to Internet applications may vary significantly, the working set often does not.

For our evaluation, we built a trace-driven simulator and focus on a representative Internet service. We simulate the major pieces of software that are of interest, namely the service's workload, the consolidation algorithm, and the middleware for cooperative caching. We also simulate the major hardware components of the cluster (CPUs, main memory, network interfaces and switch, and disks), their utilizations, bandwidths, latencies, and power consumptions. Next, we describe these aspects of our simulator in greater detail.

5.3.1. Internet Service and Its Workload. Our representative application is a "snippet" generator that services Web search queries by returning a query-dependent summary of the search results. Each query generates a list of 10 URLs. The snippet generator scans the pages associated with these URLs and produces a text snippet for each of them. It uses the middleware to cache the pages.

We obtained a 7-day trace representing a fraction of the query traffic directed to a popular search engine (Ask.com). Due to privacy and commercial concerns, the trace only includes information about the number of queries per second. The volume of queries follows the traditional pattern of peaks during the day and troughs during the night. Weekend traffic follows a similar pattern but with lower traffic. In order to generate a complete workload, we also analyze publicly available traces that contain all submitted queries (36.39 Million) to AOL over a duration of 3 months in 2006. The distribution of object popularity follows a Zipfian distribution [Adamic 2000]. We ran a sample of AOL queries against Ask.com, downloaded the content pointed to by the URLs listed in the returned results, and computed the content size. We found a median size of 6Kbytes following a Gamma distribution. In our experiments, we run two days of the trace, corresponding to a Friday and a Saturday, as well as a few extra hours of cache warm-up time prepended.

5.3.2. Cooperative Caching Middleware. Our middleware implements the PRESS cooperative main-memory caching system [Carrera and Bianchini 2005], but modifies it to accommodate servers in a barely alive state and to resize the local caches dynamically. The goal is to reach a target cache hit ratio, while allowing energy conservation and freeing up as much memory as possible for applications. Note that the middleware cannot target an average response time, since it does not service the cache misses (as explained shortly). We assume that each application knows the average response time it wants to achieve, computes the target hit ratio based on this response time and the average cache hit/miss times, and informs the middleware about the computed target hit ratio.

Request distribution. The middleware caches application-level objects and names them using numerical ids. It maintains the location of each cached object in the *cooperative cache directory*, which is replicated at each server. When first received by the service, a client request is assigned to a server in round-robin fashion. This initial server decides whether to actually serve the request, depending on whether it caches the requested object. If it does not, it looks up the directory and forwards the request to a server that does (if one exists). If the remote server is in the BA2 state, the initial server accesses the object directly from its memory. If the remote server is overloaded, the initial server does not forward the request and replicates the object locally.

Applications interact with the middleware mainly by calling runtime routines for storing and fetching objects to/from the cooperative cache. A fetch call that misses the cache returns a flag reflecting the miss; in this case, the application is supposed to fetch or regenerate the object and store it in the cache. The middleware also provides calls for object invalidation. The middleware allows these calls to originate at any *active* server, that is, servers in a barely alive state are essentially passive "object fetch servers". The servers in the BA2 state can find objects in memory because the network interface processor shares the object addresses with the host processor. Invalidating an object cached by a barely alive server works fine, because when the barely alive server is activated, it realizes that the object should be invalidated by contacting one of the active nodes. To prevent the loss of cache space at the barely alive servers in invalidate-intensive scenarios, they can be periodically activated, while some of the active servers can be sent to the barely alive state.

Local cache resizing. The middleware determines the local (LRU) cache sizes that are required for a target hit ratio using the stack algorithm [Mattson et al. 1970]. The key characteristic of the algorithm is that it can compute the hit ratio that would be achieved by all cache sizes using a single pass over the stream of memory accesses. The idea is to keep an "LRU stack" of memory block addresses (objects are broken up into blocks) sorted by recency of access; an access moves the corresponding block address to the top of the stack. In addition, the algorithm computes the "stack distance" between two consecutive accesses to each block. On an access, the stack distance is the number of other blocks between the current location of the accessed block and the top of the stack. The distance reflects the number of other blocks that were accessed between the current and the previous access to the block. A distance larger than the number of blocks that fit in each cache size represents a cache miss for that size.

The middleware periodically (every hour) collects the stack information from all active servers and computes the total (cooperative) cache size required by the target hit ratio. In systems that consolidate workloads and turn servers off, the middleware sets the local caches to their maximum size and informs the consolidation algorithm about the minimum number of servers (= total size divided by maximum local size) that need to remain active. In systems that use barely alive states, the middleware sets the local cache sizes to the total cache size divided by the total number of servers.

5.3.3. Consolidation Algorithm. We use a consolidation algorithm that periodically (every hour) determines how many servers should remain active while others can be transitioned to a low-power state (barely alive, Somniloquy, or off state). The behavior of the algorithm depends on the type of low-power state the system wants to use.

For systems that use a barely alive state, the number of active servers is based solely on the average utilization of the resource that is closest to saturation [Heath et al. 2005; Pinheiro et al. 2001]. As a server-wide proxy for this average utilization, we use the average number of outstanding requests divided by the maximum number of outstanding requests a server can handle efficiently given the workload. Using this metric, when the average response time increases, the utilization also increases.

When the average utilization cluster-wide is lower than the "state transition threshold", the algorithm tries to reduce the number of active servers. Its main constraint is that, after consolidation, no server shall exhibit a utilization higher than this threshold. As discussed in Section 5.1, when provisioning for potential load spikes, the algorithm adds extra active nodes to compensate for activation delays.

For systems that use Somniloquy or off states, the number of active servers is the maximum between the preceding utilization-based calculation and the hit-ratio-based minimum number of active servers described in the previous subsection.

5.3.4. Discussion. We simulate a relatively simple single-tier service to demonstrate the benefits of barely alive states and dynamic state transitions more clearly. In this service, the application data is placed in such a way that any active server can handle a cache miss by performing local disk I/O. This simplifies the consolidation algorithm and allows the system to: (1) turn off the entire CPU and all disks in the barely alive (BA2) state; or (2) transition some servers to Somniloquy or off state.

In practice, services are often more complex with multiple tiers and highly distributed datasets. In such services, the low-power states that can be used depend on the characteristics of each tier. For example, some of the tiers may involve servers that do not require much computation or disk I/O. The system can easily transition some of those servers to deep barely alive states (e.g., BA2). For the tiers that do require computation and disk I/O, a cache miss may cause the server to perform some local computation and communicate with other servers that will perform more computation and disk I/O. The system can still transition some of these servers to a barely alive state (e.g., BA5-1C-1D) and perform computation and disk I/O in that state. In contrast, the other schemes would either: (1) be unable to use low-power states at all; or (2) have to activate the servers first. Thus, the benefits of our server states would be even clearer for these tiers.

5.3.5. Simulation Methodology. We implement a detailed trace-driven simulator, which we use with our two-day trace. We model the workload using tuples of the form *(object-id, object-size, timestamp)*. The simulator takes the workload as input and implements all aspects of the caching middleware in detail. It simulates an LRU stack and hits table for each node, which it updates using the requested *object-id*. It also simulates the memory usage accurately, using the *object-size* information. The simulator also implements the consolidation algorithm in detail.

We simulate 32-node clusters by default. We provision the clusters for the peak demand of our application. Specifically, when all servers are active, the average server utilization at the peak load intensity is roughly 70%. This setting allows enough slack to handle major unexpected increases in load. We set the default state transition threshold to 85% of the 70% of the peak utilization, that is, in terms of actual utilization the threshold is: 70% * 85% = 59%. Henceforth, when we mention a state transition threshold, its value is always relative to the 70% peak load intensity.

The simulator models all the major hardware components of each server and the interconnect. We assume that a server's CPU utilization is directly proportional to the request load currently handled by the server. Disk utilizations and latencies are computed by accounting for average seek, rotational, and data transfer times. Similarly, the memory utilizations and latencies are computed by accounting for row buffer hits and misses. The interconnect performance is modeled by a TCP connection establishment time and its communication bandwidth.

Each simulated server has two Xeon CPUs (each with 4 cores), two 2GB DIMMs of DDR3 main memory, two 7200rpm disks, one 1 Gbit Ethernet network interface, and five fans. We assume that the middleware is allowed to manage 1/4 of the main memory of each server. When in BA2, many of these components can be turned off. The default performance parameters of our servers are described in Table II. The power consumptions of our servers in the active, BA2, and Somniloquy states are presented in Table III (servers that are off consume 0W). These performance and power parameters came from real datasheets and papers [Hitachi 2011; Intel 2009, 2010; Lim et al. 2008; Micron 2011]. We do not simulate PowerNap because there are

Table II. Server Performance Parameters

Component	Type	Performance
CPU	High-end	2.66 GHz Xeon
	Low-end	1.66 GHz Atom
Memory	DDR3	Row access: 35 nsec Column access: 20 nsec Row size: 2 KB Access size: 64 bytes
Disk	High-end	Avg seek time: 8.2 msec Avg rotational time: 4.2 msec Media transfer rate: 130 MB/sec
	Low-end	Avg seek time: 11 msec Avg rotational time: 4.2 msec Media transfer rate: 155.6 MB/sec
Network	Ethernet	Connection establishment: (24+19)*0.001 msec Bandwidth: 1 Gbit/sec

Table III. Server Power Consumption by Component

Component	Active	BA2	Somniloquy	Low-end
Core i7 (Xeon 5500) CPU	94-260W	18W (2)	2W	
Atom (D500) CPU				0-26W
1 Gbit/sec NI	5W	5W	6W (64MB)	5W
2 Hitachi Deskstar 7K1000	24W	4W	4W	
500MB laptop disk				10W
DRAM	12W (4GB)	12W (4GB)	0.7W (4GB)	5W (1GB)
Fans	50W (5)	10W (1)	10W (1)	10W (1)
Small embedded CPU		1W		
Power supply loss	37-53W (20%-15%)	10W (20%)	5W (20%)	5-8W (15%)
Total	222-404W	60W	28W	35-64W

very few opportunities to use it with 8 cores. Note that the CPUs still consume 18W in the BA2 state, because their memory controllers remain active. Similarly, the disks still consume 2W each, because their interfaces need to remain on even when the disks have been spun down. We compute the power consumption of each active server as a linear function of utilization between their minimum and maximum consumptions. As we can see, the power savings due to the BA2 state range from 162 to 344W, as compared to the active state.

For further comparison, we also simulate clusters built out of lower-power (and lower-performance) servers that use mobile-class processors. Researchers have argued for using such servers in services, rather than workload consolidation and server turn off [Andersen et al. 2009; Reddi et al. 2010; Lim et al. 2008]. The parameters we use for these servers are listed in Table III under the “Low-end” heading.

Defining the number of low-end servers to use in a fair comparison with our system is difficult. We simulate low-end clusters 6 times larger than the other systems for two reasons: (1) each of the high-end servers includes two processors; and (2) previous work [Reddi et al. 2010] suggested that 3x is the performance loss of low-end servers compared to single-processor high-end servers. The middleware manages 1/4 of the memory of each server (256MB); the same ratio we use for the high-performance servers. As mentioned earlier, consolidation is turned off. In summary, our low-end configuration uses 192 nodes (instead of 32 nodes in the high-performance configuration) and a total cache space of 48GB (instead of 32GB).

5.3.6. Results. In our first set of simulations, we consider the case in which the systems we study are provisioned without expecting load spikes. We first identify the maximum

Table IV. Energy Consumption and Savings without Spike Provisioning

	Weekday		Weekend-day	
System	Energy [Wh/day]	Energy Savings[%]	Energy [Wh/day]	Energy Savings[%]
Baseline	229845	0	219108	0
BA2	169522	26.2	144224	34.2
On/Off	198678	13.6	187773	14.3
Somniloquy	218875	4.8	216920	1.0
Low-end	185020	19.5	176680	19.4

hit ratio that can be achieved by our workload (all nodes active using their entire memories for object caching). We refer to this hit ratio as the baseline ratio. As our default, we set the target hit ratio for all systems to 95% of this baseline ratio. The total amount of cache space required by this target hit ratio is 26GB.

Our results show that the BA2, Somniloquy, On/Off, Low-end, and baseline systems achieve an average response time within 1–2% of 23ms during both days we study. Despite the similar performance, the energy savings achieved by these systems differ significantly. Table IV lists the energy consumption and savings with respect to the baseline system. As we can see from the table, in this scenario, the BA2 system achieves at least twice the energy savings of the On/Off system. The reason is that the On/Off system needs to always maintain a relatively large number of active servers to satisfy the target hit ratio. The BA2 system, on the other hand, keeps only as many active servers as necessary to service the current offered load; it transitions the other servers to the BA2 state. The top graphs in Figure 4 show the number of active servers (left) and average power consumption (right) of these systems over time.

The advantage of the BA2 system is even more pronounced when we compare it against the Somniloquy system. In Somniloquy state, a server can only store 64MB, which is only a small contribution to the global cache. For this reason, the Somniloquy system needs to keep many more active servers than the BA2 system. In fact, the former system keeps only slightly fewer active servers than the On/Off system. Again, the top graphs in Figure 4 illustrate these behaviors.

The Low-end system achieves the second best energy savings; 7–15% lower savings than the BA2 system. Figure 4 does not show the behavior of the Low-end system because it keeps all servers active during the entire execution.

In our second set of simulations, we consider the more realistic case in which load spikes may occur and must be provisioned for. In this case, the On/Off system is provisioned to keep some additional active nodes, so that the spikes can be handled without performance degradation. Specifically, the additional provisioning translates into 5 extra active nodes over time. In contrast, the BA2 system activates servers much faster so it only needs 3 additional active nodes. The Somniloquy system also transitions fast because the caching middleware does not perform store operations to servers that are in a low-power state. For this reason, it also only needs 3 extra active servers.

Table V summarizes the results assuming spike provisioning; the Baseline and Low-end systems behave as in Table IV, as they never transition power states and, thus, do not require extra active servers. The bottom graphs of Figure 4 show the number of active nodes and power consumption over time, assuming spike provisioning. We observe that the penalty of the extra active servers hurts the On/Off system significantly more than the BA2 and Somniloquy systems. Moreover, we can see that the additional active server is enough to cause an increase in energy consumption for the Somniloquy system compared to the baseline. In contrast, the energy savings of the BA2 system decrease by only 1%.

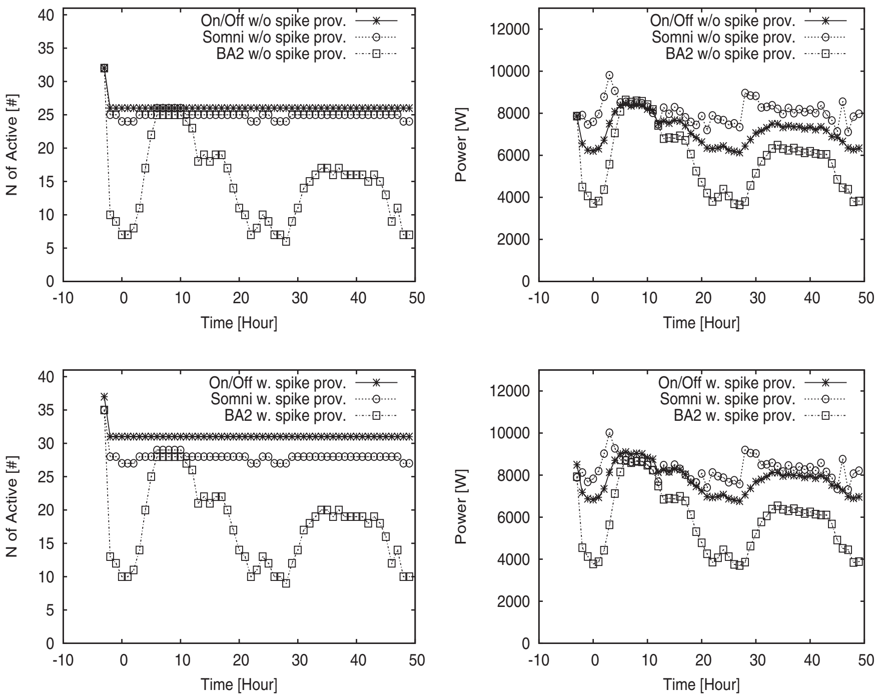

Fig. 4. Number of active servers (left) and power (right) over time, without (first row) and with (second row) spike provisioning.

Table V. Energy Consumption and Savings with Spike Provisioning

System	Weekday		Weekend-day	
	Energy [Wh/day]	Energy Savings[%]	Energy [Wh/day]	Energy Savings[%]
Base	229845	0	219108	0
BA2	171142	25.5	145844	33.4
On/Off	215688	6.2	204783	6.5
Somniloquy	224309	2.4	222484	−1.5

Note that the Somniloquy results are substantially worse than in Agarwal et al. [2009], where it was mainly used to keep idle desktop machines network-connected. As we discuss earlier, for data-intensive Internet services, Somniloquy would require much larger memory to be competitive. For services that include frequent writes, the fact that the writes would not be performed in place in Somniloquy is also a problem.

5.3.7. Sensitivity Analysis. In this section, we evaluate the sensitivity of our results to three key parameters: the state transition threshold for consolidation; the ratio of active and barely alive (BA2) powers; and the range of load intensities of the workload. Unless otherwise stated, we assume the scenario with provisioning for load spikes.

State transition threshold. Recall that this threshold determines how much the systems consolidate; the lower the threshold, the more machines are kept active. Table VI shows the results for threshold values ranging from 50–85%. Recall that 85% is our default threshold setting. As one would expect, the table shows that the energy savings that can be achieved by the BA2 system decrease significantly, as we decrease the threshold. Nevertheless, even at the most aggressive setting (50%), the BA2 system still achieves significant energy savings (11% and 18%). The small energy savings from

Table VI. Sensitivity to State Transition Threshold with Spike Provisioning

	Weekday		Weekend-day	
System (transition threshold)	Energy [Wh/day]	Energy Savings[%]	Energy [Wh/day]	Energy Savings[%]
Base	229845	0	219108	0
BA2(85%)	171142	25.5	145844	33.4
BA2(70%)	184926	19.5	156823	28.4
BA2(50%)	203984	11.3	180025	17.8
On/Off(85%)	215688	6.2	204783	6.5
On/Off(70%)	221841	3.5	204783	6.5
On/Off(50%)	231440	−0.7	209371	4.4
Somniloquy(85%)	224309	2.4	222484	−1.5
Somniloquy(70%)	221146	3.8	209928	4.2
Somniloquy(50%)	222199	3.3	201064	8.2

the On/Off system also degrade with lower thresholds. In fact, for the lowest threshold, the On/Off system actually consumes more energy than the baseline on Friday. The Somniloquy results are more interesting in that decreasing the threshold sometimes increases energy savings. The reason is that a lower threshold enables the Somniloquy system to use more memory for caching (since there are more active servers), improving its cache hit ratio.

Target hit ratio. The target hit ratio is the main performance parameter in our systems: the higher the target, the lower the response time of the service. We consider three target hit ratios: 95% (our default), 90%, and 85% of the maximum achievable hit ratio. These hit ratios require 26GB, 22GB, and 17GB of main-memory cache in the On/Off system. For these simulations, we keep the state transition threshold at its default value (85%).

We again find that the systems achieve average response times within a few percent of each other for each target hit ratio. In terms of energy, decreasing the target hit ratio affects the On/Off and Somniloquy systems more strongly than the BA2 system. Specifically, for a system without spike provisioning, the energy savings of the BA2 system decrease slightly from 26% to 23%, when we decrease the target hit ratio from 95% to 85%. The decrease in energy savings is a result of slightly higher server and disk utilizations. In contrast, the energy savings of the On/Off system increase from 14% to 26%, with the same change in target hit ratio. The reason for such a large improvement is that the On/Off system requires many fewer active servers with the lower target. The Somniloquy system also benefits from the lower target hit ratio, but to a smaller extent than the On/Off system. Under spike provisioning, the BA2 energy savings surpass those of its counterparts even at the 85% target hit ratio. These results illustrate that the advantage of the BA2 systems is greater when the service's performance requirements are more stringent.

Ratio of active and BA2 powers. Under our hardware assumptions, this ratio is roughly 7:1. We also studied ratios of 13:1, 3:1, and 1:1, under our default state transition threshold. For the weekday, these ratios produce energy savings of 31.6%, 15.6%, and −21.8%, respectively. For the weekend day, the savings are 41.1%, 20.4%, and −27.9%, respectively. These results show that the BA2 system can conserve substantial energy even at a low 3:1 ratio.

Range of load intensities. Finally, we investigate the impact of the difference in load intensity between the peak and valley of the workload, assuming our default state transition threshold and power parameters. Specifically, we scale down the difference between these load intensities by up to a factor of 4. As expected, the lower the load variation, the lower the energy savings that can be achieved. Nevertheless, the BA2 energy savings reach 10.5% on a weekday and 13.7% on a weekend day, even when the load variation is reduced by a factor of 4.

6. COMBINING MULTIPLE LOW-POWER STATES

So far, we have considered systems that leverage a single low-power state for energy conservation. In this section, we propose a “mixed” system that combines off and BA2 states in the context of our cooperative caching middleware. The motivation is that the BA2 system could potentially turn servers off to conserve even more energy, when they are not needed to achieve the target hit ratio. In addition, we study the trade-off between response time (represented by target hit ratio), state transition threshold, and energy savings.

6.1. Mixed System: Off + BA2

The BA2 system we have discussed so far has an important characteristic: it minimizes the number of active servers; the number of such servers is the minimum required by the offered request load. However, the BA2 system may not require all the other nodes to be in BA2 state; it may be possible to satisfy the target hit ratio with fewer servers, and turn the others completely off. This is what our mixed system does.

Specifically, the mixed system activates as many servers as directed by the consolidation algorithm for a BA2 system. However, instead of using the local cache resizing approach of the BA2 system, it uses that of the On/Off system. In other words, instead of resizing the local caches of all servers so that their sum is equal to the total required cache size, it resizes them to their maximum size and defines the minimum number of servers that is needed to reach the total required cache size. If this minimum number is larger than the number of active servers computed by the consolidation algorithm, the difference between them is the number of servers that will be transitioned to the BA2 state. The system transitions the other servers to the off state.

6.2. Results

Figure 5 shows the number of active and BA2 servers in our mixed system during the two days without spike provisioning. Since the target hit ratio (95% of the highest achievable ratio) is fixed, the total number of servers in the active or BA2 state stays constant (26) throughout the two days. The other servers stay in the off state.

This behavior enables the mixed system to achieve higher energy savings than its counterparts. In more detail, the mixed system achieves energy savings of 30% for Friday and 38% for Saturday, both with respect to the baseline system. Recall that the BA2, On/Off, and Somniloquy systems achieve energy savings of 26%, 14%, and 5% for Friday, and 34%, 14%, and 1% for Saturday, respectively. These results illustrate the potential energy benefits of leveraging both BA2 and off states.

We now compare the mixed system to the BA2 and On/Off systems, as a function of the performance level (i.e., target hit ratio) and the state transition threshold. We do not include the Somniloquy system because it behaves substantially worse than the other systems. Table VII presents the results broken down by day. Again, the energy savings are computed with respect to the baseline system.

In the mixed system, for a fixed target hit ratio, a higher state transition threshold enables more servers to be in the BA2 state, instead of the active state. Thus, increasing the threshold increases the energy savings. For a fixed state transition threshold, a higher target hit ratio requires more servers to be in the BA2 state, instead of the off state. Since BA2 consumes little power and the number of servers in this situation is fairly small, the impact of varying the target hit ratio is very small in this system. This result suggests that the mixed system is resilient regardless of the desired performance.

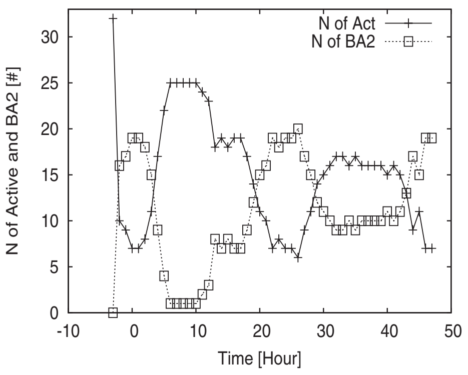

Fig. 5. Number of active and BA2 servers in the mixed system (without spike provisioning).

Table VII. Energy Savings, as a Function of State Transition Threshold, Performance Level, and Day

	Weekday	Weekend-day
System (transition threshold)	Energy Savings[%]	Energy Savings[%]
Target hit ratio = 95% of max		
BA2(85%)	26.2	34.2
BA2(70%)	20.2	29.2
BA2(50%)	12.0	18.6
On/Off(85%)	13.6	14.3
On/Off(70%)	10.9	14.3
On/Off(50%)	6.7	12.2
Mixed(85%)	30.3	38.4
Mixed(70%)	23.6	33.4
Mixed(50%)	14.0	22.4
Target hit ratio = 90% of max		
BA2(85%)	25	33.1
BA2(70%)	18.8	28.0
BA2(50%)	10.8	17.0
On/Off(85%)	20.3	23.8
On/Off(70%)	16.6	23.8
On/Off(50%)	9.7	16.4
Mixed(85%)	31.2	40.3
Mixed(70%)	23.8	35.2
Mixed(50%)	13.9	22.0
Target hit ratio = 85% of max		
BA2(85%)	23.4	31.8
BA2(70%)	17.4	26.9
BA2(50%)	9.7	15.5
On/Off(85%)	26.3	34.9
On/Off(70%)	20.5	30.9
On/Off(50%)	12.5	19.9
Mixed(85%)	31.4	42.4
Mixed(70%)	23.5	36.3
Mixed(50%)	13.8	21.5

We study the energy behavior of the BA2 system for a fixed target hit ratio and varying state transition threshold in the previous section. For a fixed state transition threshold, a higher target hit ratio increases the energy savings slightly because server (and disk) utilization decrease. Again, we consider the energy behavior of the On/Off system for a fixed target hit ratio and varying state transition threshold in the previous section. For a fixed state transition threshold, a higher target hit ratio decreases the energy savings because more servers have to stay active.

Summary. Overall, these results demonstrate that the mixed system consistently conserves more energy than the BA2 and On/Off systems. Compared to the BA2 system, the advantage of the mixed system is most pronounced at low target hit ratios and high state transition thresholds. Given these results, the mixed system is the clear choice for services that can accept higher response times or want to conserve more energy.

Compared to the On/Off system, the advantage of the mixed system is most pronounced at high target hit ratios and high state transition thresholds. In this comparison, the mixed system is the clear choice for services that require lower response times or want to conserve more energy.

7. CONCLUSION

In this article, we introduced the barely alive family of low-power server states. We compared the family to conventional on-off consolidation, other low-power server schemes, and low-end servers. We found that the ability to access memory while in a low-power state has important advantages for both keeping data current and for cooperative caching. Our study of an Internet service workload with cooperative caching showed that conserving energy by using only a barely alive state can save significant energy, up to 34%. Energy savings can be even higher, up to 38%, when the service may transition servers to either a barely alive and off states.

REFERENCES

Adamic, L. 2000. Zipf, power-laws, and pareto – A ranking tutorial. Tech. rep., HP Labs.

Agarwal, Y., Hodges, S., Chandra, R., Scott, J., Bahl, P., and Gupta, R. 2009. Somniloquy: Augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 365–380.

Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. 2009. FAWN: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*. ACM, New York, 1–14.

Barroso, L. A. and Hölzle, U. 2007. The case for energy-proportional computing. *Comput. 40*, 33–37.

Carrera, E. V. and Bianchini, R. 2005. PRESS: A clustered server based on user-level communication. *IEEE Trans. Parallel Distrib. Syst. 16*, 385–395.

Carrera, E. V., Pinheiro, E., and Bianchini, R. 2003. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS'03)*. ACM, New York, 86–97.

Chase, J., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*. ACM, New York, 103–116.

Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., and Zhao, F. 2008. Energy-Aware server provisioning and load dispatching for connection-intensive Internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, 337–350.

Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., and Gautam, N. 2005. Managing server energy and operational costs in hosting centers. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.* ACM, New York, 303–314.

Dormando. 2011. Memcached. http://memcached.org.

Elnozahy, M., Kistler, M., and Rajamony, R. 2003. Energy-Efficient server clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computer Systems (PACS'02)*. Springer, 179–197.

Fan, X., Weber, W.-D., and Barroso, L. A. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, 13–23.

Fitzpatrick, B. 2004. Distributed caching with memcached. *Linux J.*

Heath, T., Diniz, B., Carrera, E. V., Meira Jr., W., and Bianchini, R. 2005. Energy conservation in heterogeneous server clusters. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'05)*. ACM, New York, 186–195.

Hitachi. 2011. Deskstar 7k1000 specification sheet. http://www.hgst.com/tech/techlib.nsf/techdocs/D70FC3A0F32161868625747B00832876/$file/Deskstar_7K1000.B_DS.pdf.

Intel. 2009. Intel Xeon processor 5500 series datasheet, volume 1. http://www.intel.com/content/www/us/en/processors/xeon-5500-vol-1-datasheet.html.

Intel. 2010. Intel Atom processor d400 and d500 series datasheet, volume 1. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/atom-d400-d500-vol-1-datasheet.pdf.

Janapa Reddi, V., Lee, B. C., Chilimbi, T., and Vaid, K. 2010. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, 314–325.

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., and Pu, C. 2009. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Proceedings of the ACM/IFIP/USENIX 10th International Conference on Middleware (Middleware'09)*. Springer, 163–183.

Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., and Reinhardt, S. 2008. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*. IEEE Computer Society, Los Alamitos,CA, 315–326.

Lim, K., Chang, J., Mudge, T., Ranganathan, P., Reinhardt, S., and Wenisch, T. F. 2009. Disaggregated memory for expansion and sharing in blade servers. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, 267–278.

Liu, J., Wu, J., and Panda, D. K. 2003. High performance rdma-based mpi implementation over infiniband. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS'03)*. ACM, New York, 295–304.

Mattson, R. L., Gecsei, J., Slutz, D. R., and Traiger, I. L. 1970. Evaluation techniques for storage hierarchies. *IBM Syst. J. 9*, 78–117.

Miesner, D., Gold, B. T., and Wenisch, T. F. 2009. PowerNap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*. ACM, New York, 205–216.

Micron. 2011. System power calculators. http://www.micron.com/support/dram/power_calc.html.

Myricom. 2009. Myrinet. http://www.myri.com/myrinet.

Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., and Nahum, E. 1998. Locality-Aware request distribution in cluster-based network servers. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'VIII)*. ACM, New York, 205–216.

Pinheiro, E., Bianchini, R., Carrera, E. V., and Heath, T. 2001. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'01)*.

Rajamani, K. and Lefurgy, C. 2003. On evaluating request-distribution schemes for saving energy in server clusters. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software.* IEEE Computer Society, Los Alamitos, CA, 111–122.

Shimpi, A. L. 2008. *Nehalem: The Unwritten Chapters*. AnandTech.

Weddle, C., Oldham, M., Qian, J., Wang, A.-I. A., Reiher, P., and Kuenning, G. 2007. PARAID: A gear-shifting power-aware RAID. *ACM Trans. Storage 3.*

Received June 2011; revised September 2011; accepted October 2011