In Proc. IEEE International Green Computing Conference, Chicago, IL. 2010.

A Light-weight Approach to Reducing Energy Management Delays in Disks*

Guanying Wang†, Ali R. Butt†, Chris Gniady‡, Puranjoy Bhattacharjee†
†Department of Computer Science
Virginia Tech
Blacksburg, USA
Email: {wanggy, butta, puran}@cs.vt.edu
‡Department of Computer Science
University of Arizona
Tucson, USA
Email: gniady@cs.arizona.edu

***Abstract*—Today's enterprise computing systems routinely employ a large number of computers for tasks ranging from supporting daily business operations to mission-critical back-end applications. These computers consume a lot of energy whose monetary cost accounts for a significant portion of an enterprise's operating budget. Consequently, enterprises employ energy saving techniques such as turning machines off overnight and dynamic energy management during the business hours. Unfortunately, dynamic energy management, especially that for disks, introduces delays when an accessed disk is in a low power state and needs to be brought into an active state. Existing techniques mainly focus on reducing energy consumption and do not take advantage of enterprise-wide resources to mitigate the associated delays. Thus, systems designers are faced with a critical trade-off: saving energy reduces operating costs but may increase the delays exposed to the users, conversely, reducing access latencies and making the system more responsive may preclude energy management techniques. In this paper, we propose System-wide Alternative Retrieval of Data (SARD) that exploits the large number of machines in an enterprise environment to transparently retrieve binaries from other nodes, thus avoiding access delays when the local disk is in a low power mode. SARD uses a software-based approach to reduce spin-up delays while eliminating the need for major operating system changes, custom buffering, or shared memory infrastructure. The main goal of SARD is not to increase energy savings, rather reduce delays associated with energy management techniques, which will encourage users to utilize energy management techniques more frequently and realize the energy savings. Our evaluation of SARD using trace-driven simulations as well as an actual implementation in a real system shows over 71% average reduction in delays associated with energy management. Moreover, SARD achieves an additional 5.1% average reduction in energy consumption for typical desktop applications compared to the widely-used timeout-based disk energy management.**

***Keywords*-Spin-up delay reduction, disk energy management, peer memory sharing;**

I. Introduction

Research on energy conservation has traditionally been focused on battery-operated devices. However, recent works have also highlighted the positive financial and environmental implications of energy conservation for stand-alone servers and workstations [2]–[5]. For example, large organizations often require shutting down workstations, unneeded servers and cooling systems overnight [6] to reduce energy costs. Setups such as academic institutions and businesses, where users frequently work remotely and at all hours, also employ dynamic energy management.

Dynamic management saves energy by identifying periods of inactivity for a device, and then keeping the device in a low-power state during such periods. Accurately predicting such idle periods [7], [8] is critical for energy reduction and minimization of exposed delays. However, these mechanisms still expose powering-on delays (e.g., disk spin-up delays), even if the predictions are correct and provide energy savings. Delays can significantly impact system performance, irritate users, and also reduce the energy savings since the system has to operate longer to satisfy user requests. Furthermore, excessive delays may irritate users to the point where they simply disable energy management techniques. *Therefore, the challenge lies in realizing the energy savings, by keeping the system powered down for as long as possible, yet reducing the performance impact associated with energy management delays, e.g., response latencies on powering the device up when it is needed.*

Current approaches for reducing energy management delays rely on predicting when I/O requests will arrive and powering-on the device ahead of time [9]. Alternatively, energy management delays can be reduced by utilizing surrogate sources that may be available [10]. The implication here is that the requests destined for a given device are somehow satisfied by a lower-power and higher-performance alternative source. Thus, the delays associated with energy management are avoided.

In this paper, we focus on reducing disk energy management delays that are significantly longer than any other system component due to disks containing mechanical platters that requires significant amount of time to spin up from a low-power mode. Moreover, disks are significant energy consumers [11]–[13]; and disk energy management, e.g., shutting down idle disks [7], is a common practice present on almost every system in some form. Subsequently, we explore alternative ways of satisfying I/O requests destined for a typical desktop or workstation disk in low-power mode

*This paper is an extended version of a poster that appeared in IEEE/ACM MASCOTS 2009 [1].

in enterprise environments. *The goal of this work is to reduce the spin-up delays by using existing resources present in such environments*. While hiding latency might not seem directly related to saving energy, it is crucial in enabling higher rate of adoption of power-saving techniques. Servicing I/Os from alternate sources provides opportunities for keeping the local disks in low-power mode, and may reduce energy consumption as an additional bonus.

To avoid spinning up the disk on arrival of user requests, and subsequently exposing spin-up delays to the users, we exploit the arrangement of local disks/file servers adopted in enterprise environments. Instead of always going to the local disk, or the centralized file server, we present System-wide Alternative Retrieval of Data (SARD) to retrieve application binaries from other workstations that are loosely arranged in a peer-to-peer (p2p) network. The key observation in SARD is that computers in enterprise environments are mostly uniformly configured to simplify system maintenance. As a result, the application binaries are identical across many peers, which allows sharing of the application binaries among them.

Our approach shares the goal of serving requests from peer nodes with cooperative caching [14], [15], however, we stress that SARD is *unique* in its goal of reducing delays associated with energy management using loosely-connected peers, without additional hardware or extensive software modifications.

SARD is designed in such a way that it: (1) does not require any custom buffering or shared memory infrastructure; (2) does not interfere with energy management of other systems; (3) does not require additional hardware resources; (4) requires few kernel modifications; and (5) allows participants to be loosely coupled and free to leave and join the system. While an extreme alternative is to have diskless workstations with all requests serviced by a central file server, this solution is not scalable and may require expensive hardware to support large enterprise environments. SARD utilizes existing resources by transparently locating the workstation with requested binaries in the memory and transmits them to the machine that requested them. We emphasize that we exploit standard virtual memory mechanisms, and do not require custom buffering that can increase memory pressure in the systems, or shared virtual memory that increases overheads of memory systems. Furthermore, our p2p approach does not require fixed configurations and does not place any constraints on peer membership in the system. The individual machines can leave and join the system freely, significantly reducing system management that more tightly coupled systems, such as shared virtual memory, would require. Finally, our p2p infrastructure aims to select peers that will not be impacted by serving SARD requests.

The resulting design provides a low-overhead approach to minimizing energy consumption in enterprise environments. Our evaluation of SARD using both a trace-driven simulations and an actual implementation in a real system shows that SARD can provide over 71% average reduction in delays associated with energy management, while achieving an additional 5.1% average reduction in energy consumption for typical desktop applications compared to the widely-used timeout-based disk energy management.

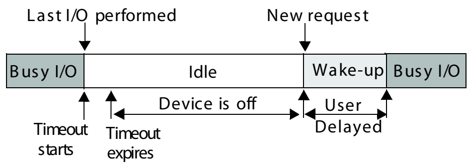

Figure 1. Anatomy of a disk idle period.

We note that the impact of SARD may be affected by the emergence of newer types of disks and non-mechanical storage such as Flash-based Solid State Drives (SSDs), as they have the potential to reduce or eliminate spin-up delays associated with current mechanical disks. In addition, SSDs consume much lesser energy compared to mechanical disks, and elaborate disk energy management schemes might not even be required in scenarios where the primary mode of storage are SSDs. However, the cost (i.e., $/GB) of SSDs remains orders of magnitude higher than that of mechanical disks. Thus, we envision that mechanical disks will remain the main storage technology over SSDs in the near and medium-term future. Consequently, making techniques such as SARD useful for the foreseeable future.

The rest of the paper is organized as follows. Section II discuss the observations and opportunities in enterprise environments that motivate this work. Section III provides the design of SARD. Section IV presents an evaluation of SARD. Section V discusses the related work. Finally, Section VI concludes the paper.

II. Opportunities in Enterprise Environments

In this section, we present several observations about desktop and workstations in large-scale enterprise environments, which serve as the key enablers for SARD.

Energy management is prevalent: Large enterprises are actively pursuing energy management of employees' workstations, since energy savings can translate into monetary savings. Users are encouraged to power down their monitors and computers once they leave work. Remote wake-up technologies such as Wake-on-LAN [16] are employed to power-on machines if necessary. After-hours management tasks such as software updates and virus checks are scheduled so that machines can remain off for most of the time they are not in active use. Furthermore, dynamic energy management is enabled to reduce energy consumption during work hours.

A popular technique for saving energy of disks is to shut the disk down after a period of idleness, as shown in Figure 1. After each request, a timer is started and the device is shut down when the timer exceeds a preset timeout interval. Any disk activity occurring during the timeout period resets the timer and prevents the disk from switching off. If the disk is powered down, it remains so until a new I/O request arrives, at that time the disk has to be fully powered up for servicing the request, which typically expose multi-second delays to the users. Frequent shutdowns/spin-ups can render energy management useless as disks use larger amount of energy to spin up, and can also shorten disk lifetime. Moreover, delays that are noticeable by users run the risk that users will switch energy management off, thus foregoing any and all energy savings.

The disk start-up delays following a spin-down due to energy management, and the associated latency observed by users, are quite significant — on the order of up to 10 seconds [9]. Thus, our strategy of retrieving binaries from peers across the network instead of from a local disk (in standby mode) reduces the delay experienced by users when employing disk energy management.

Similarly maintained systems: Managing large computing infrastructure is difficult and require large support staff. To simplify the management, especially in academic setups, the systems are kept mostly uniform: they run the same operating system and set of applications, usually on similar hardware. The similarity in hardware is also due to the fact that it is acquired in batches, e.g., from an equipment grant or industry donation. In many cases, disk duplication is used for quickly bringing new workstations on line instead of slow and error-prone individual installation and configuration. This uniformity increases the opportunity to share binaries among the participants, and individual workstations do not have to depend only on local disks. Heterogeneous systems – desktop vs. laptops, slower older vs. new resources, etc. – do not preclude binary sharing, however they limit sharing to the binaries that are identical among sharers.

User data on central file servers: Persistence is crucial for user data. Unlike system binaries, user data cannot be simply reconstructed. Thus, it is periodically backed up to protect against failures. To simplify the backup process and to provide users transparent access to data from any workstation, many setups provide central storage for user data, which is more reliable and cost effective than backing up individual systems. As a result, individual workstations usually do not contain any permanent user data. The local disks are typically utilized to boot OS and supporting temporary scratch space. Consequently, the data that may be written to the local disk is temporary, e.g., intermediate files, system logs, metadata etc. Given that preserving temporary data is not crucial, we can elongate the timer between the invocations of daemons that flush the buffer cache to the disk (e.g., the daemon *pdflush* in case of Linux) to further increase energy efficiency.

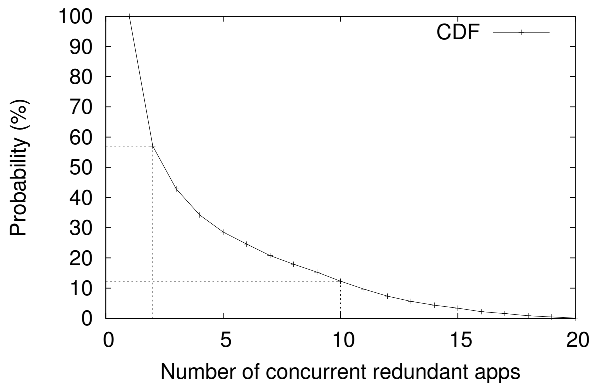

Figure 2. CDF of concurrent redundant applications.

Similar system usage: The applicability of SARD is dependent on how often different machines in an enterprise use the same binary. To investigate this, we conducted a study using 20 of our departmental machines used by students for course work, projects, and typical desktop use. For a period of 13 days, we recorded the applications that are running on each machine every 5 seconds. Next, we determined how often different machines run the same applications.

Figure 2 shows the cumulative distribution of the number of times when different machines are running the same applications. For the studied environment, 57% of the time two or more machines were running an application concurrently, and 12% of the time more than 10 copies of an application were running concurrently on different machines. Assuming a similar application usage distribution, and extrapolating these results[1] show that in a medium-scale setup with 107 machines more than 99% of the time an application will be running on at least two machines and thus can be serviced remotely. Consequently, our assumption about remote application availability holds for typical enterprise environments, and they can benefit from SARD.

Our goal is to mitigate the energy-management related delays faced in enterprises by leveraging some of the benefits provided by the uniform nature of the environment. The above observations allow us to simplify our system design for the target environments. In this case, the workstation that wants to start a new application can get a copy of the memory image of the application from a remote machine. The disk is needed only if no other system is currently running the particular application.

[1]Analysis shows that for a setup of n ($n >> 19$) nodes, the probability that two copies of an application are running at the same time is $1 - \prod [(1 + (n-1)p_i)(1 - p_i)^{n-1}]$, where p_i is the probability that an application i ($0 < i \leq n$) is running on a node at a given time. A detailed derivation is out of scope of this paper.

Security: Sharing binaries across participants has security implications in that a compromised workstation may affect others. There are several techniques we can adopt to minimize this occurrence. For example, only allow sharing of binaries from administrator-maintained standard installation paths and machines, which are protected using standard system security. Alternatively, advanced crypto-checksums, e.g., as used in self-certifying binaries [17], for isolation of tainted binaries or in-memory modified pages can be used to prevent malicious behavior. There has been considerable work on securely sharing binary executables, e.g., by Hollingswork and Miller [18]; however a full discussion of such techniques and addressing all security concerns is beyond the scope of this paper.

III. Design

SARD is targeted at desktops and workstations in enterprise setups where all machines are centrally owned and controlled, and have similar software configurations. We adopt a low-overhead approach in our design that avoids extensive kernel modifications to support ease of implementation, maintenance, and porting to future kernel versions.

A. Overview

In SARD, all machines join a p2p overlay network, which enables them to interact with each other in a decentralized and dynamic fashion. Participants run our software that advertises their in-memory applications to others via the overlay. Advertisements enable participants to learn what applications (or parts thereof) are available in memory of peers. When an application is executed on a node, it can use the remote availability information and decide whether to retrieve the application from the local disk or remote memory. Servicing requests from remote memory helps avoid spin-up delays of powered down disks, and can also improve energy savings by keeping disks in low-power mode longer.

In the following, we first discuss how system calls are rerouted to utilize application images from peer nodes, then we discuss how actual images are shared across peer node memory, followed by a discussion of how we enable nodes to find appropriate nodes in a decentralized fashion. Finally, we present a number of heuristics for when to use remote image retrieval versus local disk access. We note that although SARD can potentially reduce disk spin-up/down cycles, which reduces mechanical wear and tear and may improve expected lifetime, such affects are unlikely to affect the disk's useful lifetime (often much smaller than expected lifetime), and thus are not evaluated.

The impact of SARD is evident by the observation that energy management delays are crucial and all energy management schemes try to avoid unnecessary spin-ups exactly for this reason. Our approach is no different, except that it makes reducing such delays the main objective to encourage adoption of energy management techniques. As stated earlier, up to 10s of delay per disk spin-up following an energy-related spin-down can be avoided using SARD.

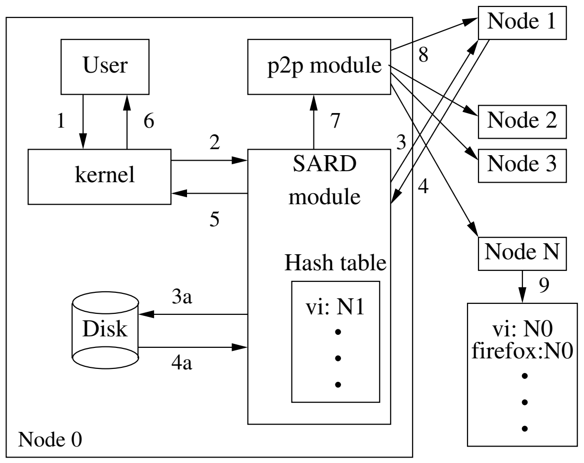

Figure 3. SARD Architecture.

B. Rerouting System Calls

For ease of explanation, we assume that an entire binary is available in memory. In reality, all modern operating systems load portions of binaries on demand, and SARD only advertises those in-memory portions. When needed, SARD can retrieve required portions from more than one remote location. As discussed in Section II, the uniformity and scale of the target environment make finding all the required portions remotely in the setup quite likely.

Figure 3 shows the architecture of SARD. The only kernel modification is to intercept and reroute disk I/O requests to the SARD module. The module consists of a UDP server, a `/proc` entry, a hash table, and interfaces required to route the intercepted I/O calls. The hash table contains information about file availability on remote nodes. The content of the hash table is provided by the p2p system (described later). The `/proc` entry (`/proc/SARD`) is used to communicate the files currently available in local memory to the p2p system for advertising to remote nodes.

After intercepting an I/O call (in the `read_pages()` function of standard Linux kernel), SARD checks the hash table to determine alternative sources for serving it. If a remote source is found, a UDP message requesting the image is sent to that node. The corresponding SARD UDP server on the remote node receives and serves the request. Once a reply containing the requested image is received back at the requester, the image is returned to the kernel just as if the request was serviced from the local disk.

Figure 3 also shows how an example call is serviced by SARD. Consider the case where a user wants to load *vi*. The user types *vi* at the command path and the *execv* initiates execution by entering the kernel (1) and mapping the image of *vi* into virtual memory. SARD intercepts the I/O requests to the disk (2) and looks up *vi* in the hash table. SARD finds *vi* in its table and that $Node$ 1 has the image in memory. Then, SARD sends out a UDP packet to $Node$ 1 with a request for *vi*'s image (3). The UDP server at $Node$ 1 retrieves the image from local memory and sends it back (4) to SARD at $Node$ 0. Once the reply is received, SARD on $Node$ 0 replies to the kernel I/O request (5). In this case, a disk access is avoided. In case SARD cannot find the requested image in memory of any known remote machine, it routes the request to the disk for servicing (3a, 4a), similarly as in the original kernel.

Additionally, after loading the pages, our module invokes the p2p module (in user space) (7), and sends out a broadcast message (8), announcing to everyone in the overlay that the node (Node 0 in this case) has the application image available in memory, and is willing to share. Other nodes then save this information (9) for later user.

C. Retrieving Images from Memory

The SARD module also retrieves application images from virtual memory on the host machine for servicing requests for specific parts of binaries from remote machines. To ensure that remote request information is portable across remote machines, the inode number in the I/O request destined for the disk is converted into a filename and path, and this information is sent to a remote machine instead of the inode number.

The SARD module first uses the filename and path information in the remote request to determine the local inode number corresponding to the requested application. It then uses the standard Linux function `kmap_atomic()` to determine the virtual address of the memory where the requested offset of the inode is stored. The contents of the memory are then sent to the requester via a UDP packet in granularity of 4 KB (the size of typical Linux memory pages). Given that the application was originally advertised as being in memory, this remote request is expected to be successful. However, if the requested image is not found in memory, a failure is returned. Upon receipt of the failure, the requester can either attempt to retrieve the image from other known remote locations, or go to the local disk as appropriate.

SARD does not share meta data about page cache across peers, and does not assume that the virtual addresses for a given application image are the same across nodes. Instead, only the contents of the page are read from memory and sent to the requester, just as the contents of a block are read from disk in a standard I/O. The memory management of the images is not tampered with and remains as in the original kernel. Once a requester receives a reply, it reads the UDP packet, copies the contents of the reply to an appropriate (already allocated) page and marks the page as ready. This is similar to what the intercepted `read_pages()` function does when reading data pages from the disk. Moreover, we target application binaries that are read only; therefore, SARD does not have to be concerned with concurrent page modifications.

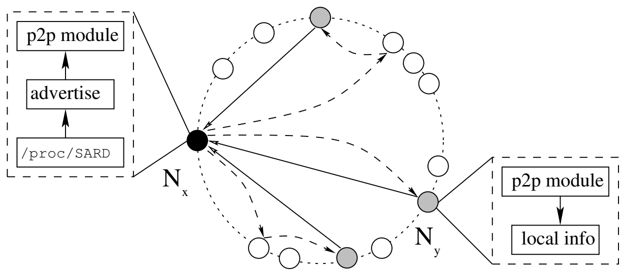

Figure 4. P2P-based available application image discovery.

D. Decentralized Application Discovery

SARD must discover and select peer nodes (N_p's) that have a given application already loaded in memory. To this end, we are faced with several choices. One solution is to have a centralized server that collects application availability information from all nodes, and uses it to assist nodes in locating alternative remote sources for I/O. The main drawback of this approach is that a special service, either on a dedicated node or on one of the central system servers, has to be maintained. Using a dedicated node is contrary to our goal of energy efficiency, while running the service on the central server may result in degraded performance of the other system services. Moreover, we want to allow the participants to be loosely connected and free to leave and join SARD at will. Another solution, which we adopt, is to utilize a p2p model that allows the participants to directly interact with each and discover binaries in a decentralized manner. This approach is motivated by recent successful application of p2p networks to building robust and scalable systems [19]–[23].

Structured p2p overlay networks [24] effectively implement scalable and fault-tolerant *distributed hash tables* (DHTs), which allow data to be inserted without knowing a-priori where it will be stored, and requests for data to be routed without requiring any knowledge of where the corresponding data items are stored. The functions provided by DHTs allow for *discovering* N_p's in a decentralized manner. We have chosen to use a DHT for peer node discovery because it avoids the issue of designating a single entity to manage the system, and thus precludes having to provide high availability etc. for the manager node. DHTs provide an elegant, scalable, and plug-and-play solution. Thus, any latency in discovering nodes due to the use of a DHT is compensated for by the benefits provided by it. Moreover, SARD is agnostic of the p2p layer used, and works equivalently well with other systems.

All nodes in a given setup, e.g., an academic lab., join a p2p network, which enables them to reliably communicate with each other. Figure 4 illustrates the discovery process. A user mode p2p daemon runs on every node and periodically monitors `/proc/SARD` and sends out a number of advertisement messages containing information about the applications that are running on a given node N_x (e.g., black node in Figure 4). The advertisements contain the host machine name, the filename and the complete binary path (and offsets of available pages). The advertisements are destined for random destination addresses, and by virtue of the DHT abstraction provided by p2p routing [24], are received at some N_y's (gray nodes). On receiving such a message, each N_y builds local information about what applications are available on N_x, and in essence *discovers* a peer node that can serve the application remotely if needed. N_y then uses this information only locally[2]. The process is periodically repeated at all participants so nodes learn about remote application availability at some other nodes. Note that since we use a random approach, not every participant will become aware of each and every location of an application. Finally, to accommodate dynamic availability of applications, nodes discard information about discovered applications after a specified period of time and start a fresh discovery process.

While lightweight, the use of random advertisements does not guarantee that all nodes can find an alternative source for an application. Providing such guarantees in a decentralized manner is hard and possible solutions can lead to complex and time consuming discovery process, which may exceed the time it would take to spin-up the local disk. Thus, we argue that our approach optimizes for the common case, and we can always rely on the local disk when no alternative remote source is known.

SARD design also helps to minimize the effect of participant churn: (1) nodes only use the advertisements locally, so their leaving the system does not affect others; (2) advertisements from failed nodes are discarded periodically; and (3) new nodes joining the system automatically receive advertisements from others by virtue of the DHT, and can advertise to others as appropriate. This is especially useful in the target enterprise environments where energy management may switch off nodes resulting in high churn.

Finally, one drawback of using DHT's is that messages require multiple hops ($O(log(n))$ in a network with n nodes). Given the number of machines in the target environment is likely to be in the order of thousands or less, we use a full membership DHT model as advocated by the design of Amazon's Dynamo [25]. To achieve full membership we modified our overlay's node join process to add a new node's address to the routing table of all the existing participants. This allows for $O(1)$ message delivery for all messages.

[2]This is in contrast to p2p storage where a node that receives a message based on a hashed filename, becomes a contact point from where other nodes in the system can locate the file.

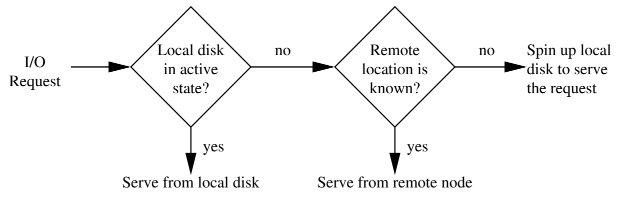
I/O Request
Local disk in active state?
no
yes
Serve from local disk
Remote location is known?
no
yes
Serve from remote node
Spin up local disk to serve the request

Figure 5. Local vs. remote retrieval heuristics in SARD.

E. Data Retrieval Heuristics

Several factors affect the decision of whether to retrieve an application from the local disk or from remote memory. We set the following intuitive set of heuristics (Figure 5) that dictate our decisions: (1) if the local disk is on, it serves the I/O request; (2) if the disk is off and no other node has the needed information, the disk is spun up and serves the request; and (3) if the disk is off and other machines have the image in memory, it is served from the remote machines.

The first heuristic implies that we should attempt to serve the requests locally when possible. This eliminates unnecessary loading of the network and remote machines. Most applications in the target environments are interactive desktop applications that involve user think time and generate long periods of inactivity, therefore the disk will be frequently in a low-power state and the two remaining heuristics will come into play. The second heuristic implies that remote machines should not retrieve needed information from the disk for a local request. This again minimizes the impact on the remote nodes and prevents potential performance degradation and simplifies our design. Alternatively, we may consider serving the I/O request by remote nodes if their disks are spinning. In this case, we would encounter much less delay than spinning the local disk. However, such an approach requires significant overhead, since the disk status of every machine would have to be communicated to all nodes. These decisions complicate the design, but are viable optimizations that we will explore in our future work. Finally, the third heuristic is the key idea that allows the local disk to remain off while I/O requests are served remotely.

IV. Evaluation

We divide the evaluation of SARD in three parts: implementation based experiments to study the impact on performance, simulation based experiments to study the impact on energy savings for typical desktop applications, and a case study of SARD under real usage conditions.

Unless otherwise noted, the experiments are performed using Dell PCs, with an Intel 2.4 GHz dual core processor, 4 GB RAM, and a high-end Seagate 250 GB hard disk. The machines are connected using 1 Gbps Ethernet.

Table I
TIMES TO RETRIEVE A FILE OF SIZE 143 MB USING DIFFERENT RETRIEVAL SCHEMES. EIGHT PAGES WERE REQUESTED AT A TIME.

Retrieval Scheme	time(s)
Disk	1.85
`get_page()`	9.60
`get_pages()`	2.54
`asyn_get_pages()`	1.40

A. Implementation Results

SARD is implemented using about 2300 lines of C code. Additional 1200 lines of Java code are used to implement the p2p advertising daemon using FreePastry [24]. Our current implementation runs on Linux kernel 2.6.

The kernel components are implemented in two parts: a kernel patch and a module. The kernel patch is kept to a minimum of adding necessary hooks into the kernel in the `read_pages` function. The rest of the functionality is realized in the kernel module. The p2p component runs as a user-level daemon and interacts with the module via `/proc/SARD`.

1) Image Retrieval Performance: In this experiment, we measure the performance of SARD at handling requests from remote memory. We use two machines as the setup for this experiment. We used the base case of reading a large file (143 MB tar file) from the local disk, and compared it with different remote image retrieval schemes. In this set of tests, one machine served as the requester, while the other provided the application image. Note that we assume that the file image is available in memory when needed for testing purposes only. As stated earlier, we expect binaries to be loaded on-demand and not available entirely in memory.

We evaluate three schemes: `get_page()`, a straw man approach, which requests a single page at a time and waits for the reply from a remote location before proceeding further; `get_pages()` that requests a group of pages in a single UDP packet, so as to amortize remote communication cost across pages; and `asyn_get_pages()` that uses call-back mechanisms to retrieve data asynchronously. As seen in Table I, reading a file under this version takes 24.3% less time than reading it from the disk. Therefore, this technique is adopted in SARD and used in rest of the evaluation.

2) Remote Binary Serving: Modern operating systems load portions of applications from disk on-demand. In a typical system that runs many different applications, on-demand accesses essentially translate to reading random pages from the disk. We model this random access behavior in the controlled experimental setting by using an application that performs random I/O.

For this purpose, we setup the PostMark [26] benchmark, which supports many knobs that essentially allow us to serve files of increasing sizes, in essence emulating on-demand random page loads of varying lengths. Note that

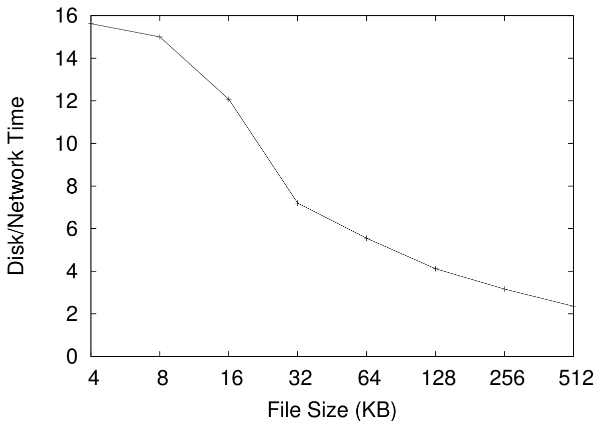

Figure 6. The ratio of local access time compared to serving the binaries remotely.

we are not using Postmark as a mail delivery system and thus not measuring mail delivery performance, rather it is being used as an emulator of binary retrieval. Consequently, we are interested in how long it takes for the system to run Postmark. For each case, we measured the time it would take to service the request locally from disk or from remote memory. Figure 6 shows the ratio of the time used for servicing a request locally compared to that served remotely. Note that for these measurements the disks were spinning and in ready state, which is the best case scenario if SARD is absent. We observe that for smaller files, the disk performance is poor compared to remote retrieval – servicing from disk takes order of magnitude longer compared to over the network. The comparative benefit from remote retrieval is somewhat reduced for larger file sizes because the time to retrieve data from the disk improves significantly with increasing file sizes, i.e. large sequential accesses. Overall, SARD provides improved performance mainly because of the fact that while random accesses have poor I/O performance for disks, the difference between random and sequential accesses is immaterial for SARD which retrieves contents from memory and does not require any disk movement.

3) Impact of SARD on Remote Machines: In the next set of experiments, we study the impact of SARD on remote node performance.

First, we determined how a node's overall performance is impacted when servicing varying rates of page requests. For this purpose, we designed a benchmark that generates a controlled number of remote page requests at one of the test machines. On the other test machine, we compiled the Linux kernel and observed the compilation time for each case as we increased the number of requests generated per second from 1 to the extreme case of 65536. Figure 7 shows the results. The horizontal line shows the average time it takes to compile the kernel on a standard setup without any remote load.

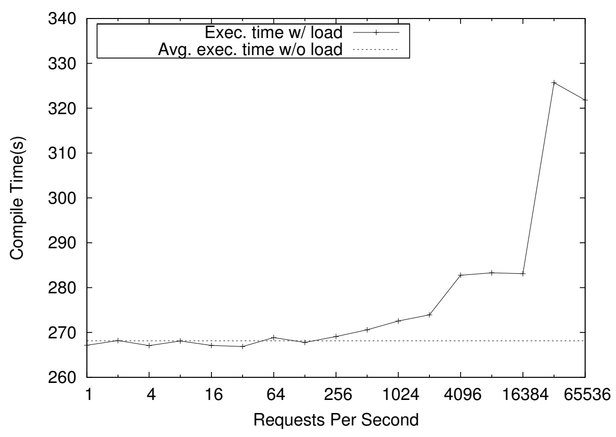

Figure 7. Impact of servicing remote memory requests.

Table II
REQUESTS-PER-SECOND FOR VARIOUS APPLICATIONS, AND THEIR ESTIMATED IMPACT ON REMOTE NODE PERFORMANCE.

Application Name	Requests per Second	Impact on Remote Node
cscope	112	0%
`make`	6.25	0%
PostMark (8 KB)	530	0.90%
PostMark (32 KB)	1073	1.63%
PostMark (128 KB)	2064	2.14%
PostMark (512 KB)	2541	2.94%

We observe that up to 256 requests per second are serviced without any observable performance degradation, and only 5.59% degradation is observed when as much as 16384 requests are serviced per second. Also note that the dip in the curve as the requests-per-second are increased to 65536 is due to lost requests from either network congestion or kernel queue overflow at such large rate.

Second, using the above information, we determined how several test applications will affect remote nodes. For this experiment, we use: a *cscope* [27] query on Linux 2.6.22.9 source code, *make* to compile the same kernel, and PostMark [26] with different file sizes. We observed the average rate of remote page requests issued by these applications. Table II shows the request rates. We then used the load impact numbers of Figure 7 to estimate the impact of the studied applications on a remote node serving the requests. Here, it is assumed that all requests from an application are serviced at a single remote node. In particular, observe that both *cscope* and `make` incur negligible overhead, and PostMark (8KB) that models on-demand application loading incurs less than 1% overhead. Furthermore, requests may be sent to multiple nodes to reduce the performance impact on individual nodes. We believe that by distributing a node's requests across multiple locations in the system, the overall rate of requests serviced at each node can be maintained within acceptable limits.

Table III
DISK ENERGY CONSUMPTION SPECIFICATIONS FOR WD2500JD.

State	
Read/Write Power	10.6W
Seek Power	13.25W
Idle Power	10W
Standby Power	1.8W
Spin-up Energy	148.5J
Shutdown Energy	6.4J
State Transition	
Spin-up time	9 sec.
Shutdown time	4 sec.

B. Simulation Results for SARD's Energy Impact

SARD can reduce the delays associated with energy management without using any additional hardware, and avoid the increase in energy consumption that additional hardware would cause. In addition to our main goal of reducing the spin-up delays exposed to the users, SARD also reduces the overall system energy consumption by servicing I/O requests from remote machines. We illustrate this by a simulation study that shows SARD's energy savings, as well as by actual system energy measurements.

1) Methodology: Detailed traces of user-interactive sessions for each application were obtained by a `strace`-based tracing tool [28] over a number of days. We used a Western Digital Caviar WD2500JD in our simulation with specifications shown in Table III. The WD2500JD has a spin-up time of about 9 seconds from a sleep state, which is common in high-speed commodity disks.

Table IV shows six desktop applications that are popular in the enterprise environments: *Mozilla* web browser, *Mplayer* music player, *Impress* presentation software, *Writer* word processor, *Calc* spreadsheet, and *Xemacs* text editor. The table also shows trace length and the details of I/O activity. Read and write requests satisfied in the buffer cache are not counted, since they do not cause disk activity. Finally, Table IV illustrates read activity in the applications by separating it into user file accesses and application file accesses. Accesses of application files dominate the read activity for interactive applications. This is true for all traced interactive applications except *Mplayer* traces which show that a majority of reads are targeting user files, reflecting the primary function of this particular application. It is clear that peer nodes mirroring common application files would meet most of the demand for file reads from the remaining applications, and that the demand for files other than application files represents a significantly smaller fraction of the total observed read requests. Finally, we assume that user files are stored on a central file server which is a common practice in an enterprise environment, as discussed in Section II. We can conclude from this that a relatively low-performance peer node can adequately meet the demands of other peers, as long as commonly accessed files are mirrored across them.

Table IV
THE NUMBER AND DURATION OF TRACES COLLECTED FOR THE STUDIED APPLICATIONS.

Appl.	Trace Length [hr]	Number of Reads	Number of Writes	Referenced [MB] Reads	Referenced [MB] Writes	User Files	Application Files
mozilla	45.97	13005	2483	66.4	19.4	17.92%	82.08%
mplayer	3.03	7980	0	32.3	0	96.37%	3.63%
impress	66.76	13907	1453	92.5	40.1	43.45%	56.55%
writer	54.19	7019	137	43.8	1.2	3.50%	96.50%
calc	53.93	5907	93	36.2	0.4	5.98%	94.02%
xemacs	92.04	23404	1062	162.8	9.4	0.15%	99.85%

Table V
NUMBER AND AVERAGE LENGTH OF APPLICATION IDLE PERIODS AS INCREASING NUMBER OF REQUESTS ARE SERVICED REMOTELY.

% of Reqs.	Mozilla		calc		Impress		Writer		Mplayer		Xemacs	
Served Locally	Idle Prds.	Length [s]	Idle Prds.	Length [s]	Idle Prds.	Length [s]	Idle Prds.	Length [s]	Idle Prds.	Length [s]	Idle Prds.	Length [s]
100	165	985	150	1283	227	1048	136	1423	4	2712	95	3477
15	102	1601	89	2170	122	1959	88	2206	4	2713	59	5604
10	88	1858	77	2511	110	2174	80	2427	4	2712	56	5906
5	82	1995	70	2763	87	2752	70	2776	4	2713	49	6751
2	58	2825	52	3724	66	3631	51	3814	4	2713	41	8070
1	49	3346	34	5701	45	5330	40	4867	2	5435	38	8708

2) Energy Consumption: Serving the I/O from remote machines increases the length of idle periods by eliminating spin-ups required to serve the I/O requests from the local disk. Table V illustrates the impact of serving I/O requests on the length of idle periods. It shows the number and average length of idle periods for varying fractions of requests served by the remote machines. The case of 100% of requests served locally illustrates the standalone workstation that serves all requests from the local disk. By serving more and more requests from other workstations the number of idle times is reduced since the idle periods are concatenated resulting in fewer and longer periods. In the case of 1% of requests served locally, the average number of idle periods is reduced by 73.2% and the average length is extended by 205.5%. Based on our study of Section II, 1% of requests served locally is a reasonable number for a medium-scale setup (e.g. with more than 107 workstations), since many will have standard applications loaded in memory. In addition, we show results for serving 2%, 5%, 10%, and 15% which may be encountered for a small number of workstations in the network.

Reduction in number of periods and lengthening the duration of the idle periods has twofold impact on energy efficiency. First, fewer number of periods indicates that there are fewer spin-ups required to serve the I/O requests resulting in lower energy spent on powering up the devices and shutting them down. Second, longer idle periods will allow the disk to remain in a power saving state also reducing energy consumption. These can be seen in Figure 8, which shows distribution of the local disk energy consumption among three categories: Busy – due to serving the I/O requests, Idle – due to waiting for more requests to arrive during timeout interval, and Power-Cycle – due to shutting down and spinning up the disk. We show numbers normalized to the case when a standard energy saving mechanism is used in a stand-alone system, i.e., with 100% requests served locally. All of the states are impacted by SARD. We first observe that energy spent serving I/O requests is not significant since most of the applications are interactive with long user think times or they are accessing user files that are mounted on a remote file server. The two largest components are power-cycle and idle energy. The average fraction of energy spent on spinning up and shutting down the disks in the case of local disk only is 69.3%. The energy is reduced as we serve more and more from remote machines and reaches the average fraction of 22.2% for the case of only 1% of requests served locally. Similarly, the time spent in idle is reduced due to fewer timeout periods encountered as we increase fraction of requests served remotely. The average fraction of energy consumed at idle is 28.8% for all requests served locally and is reduced down to 7.9% for when serving only 1% of requests locally. Moreover, compared to an always-on scheme with no energy saving mechanism, the 100% approach provides an average savings of 80.6%, which is further improved to an average of 81.7% for the case with 1% local requests.

Table VI
DELAY DUE TO DISK SPIN-UP AS MORE AND MORE REQUESTS ARE SERVICED FROM REMOTE NODE.

Local	Total Delay [s]					
%	Mozilla	Calc	Impress	Writer	Mplayer	Xemacs
100	1485	1350	2043	1224	36	855
15	918	801	1098	792	36	531
10	792	693	990	720	36	504
5	738	630	783	630	36	441
2	522	468	594	459	36	369
1	441	306	405	360	18	342

Fewer needed spin-ups result in shorter overall delays exposed to the user. For this experiment, based on our observations of the network traffic of our test departmental

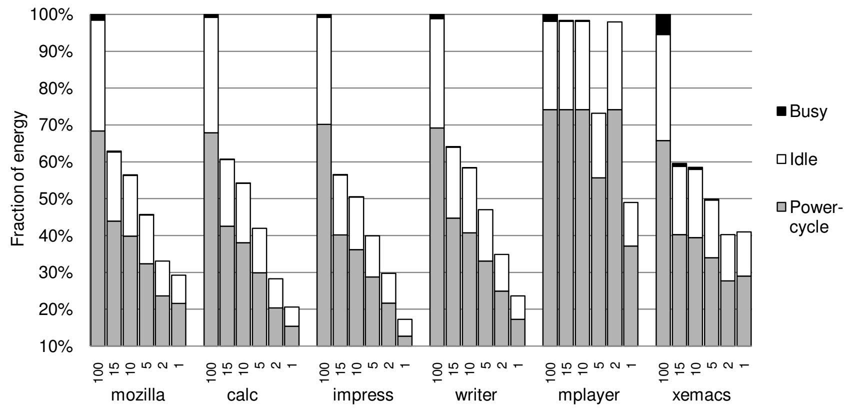

Figure 8. Breakdown of energy savings as more and more requests are serviced from remote nodes normalized to the case of a standalone system (100% local requests).

machines, we assumed network latencies of a 1 Gbps Ethernet connection with at most 20% degradation due to contention modeled randomly. Table VI illustrates the total delay exposed to the user as we vary the fraction of I/O requests served locally. The average delay across applications is reduced from 1165.5 seconds for all requests served locally to 312 seconds, i.e., a 73% reduction, when we only serve 1% of requests locally. Reduction in delay has two benefits. First, the user experience is improved since the user will see fewer lags due to disk spinning up. As a result, the user is more likely to use energy management techniques as opposed to turning the energy management off to prevent the irritating delays. Second, the shorter delays will allow the user to accomplish the task quicker, which increases the efficiency of the system.

The increase in energy efficiency of the system can be illustrated by the energy-delay product (EDP) [29]. Figure 9 shows the EDP for the studied applications, normalized to the case of a standalone system with 100% requests served locally. Each point is calculated by multiplying the total execution time (Table V) of a trace and the corresponding energy consumed (Figure 8). On average, the energy-delay product is reduced by 5.8% for the system serving only 1% of requests locally as compared to a standalone system with 100% requests serviced locally.

An observation here is that while the use of p2p overlay may lead to increased CPU energy consumption for processing the network stack, the power impact of managing the overlay is encapsulated by the overall power consumed by SARD. Therefore, we do not need to determine the finer-grained power consumption of the overlay; we can still claim that the strategy of retrieving application binary pages over the network is more efficient than that of retrieving them from the disk.

We note that, longer delays prevent users from adopting energy-saving measures. SARD can remedy this by reducing the exposed delays to the user, who will otherwise opt for no energy management: SARD reduces energy-delay product by 81.63% on average, compared to the always on case. Consequently, *SARD can potentially hasten the adaptation of energy-saving mechanisms.*

C. SARD Case Study

In our next experiment, we studied the energy saving and performance impact of SARD using 10 of our departmental machines used for typical desktop use. Each machine has a an Intel Pentium 4 3.0 GHz processor, 1 GB RAM, 40 GB hard disk, and is connected via 1 Gbps Ethernet. The network interface card (NIC) on all the participating nodes in our experiment remains in active power-up mode. Therefore, we do not experience any NIC wakeup latency when retrieving data remotely. If the NIC is put to sleep as well, techniques such as Somniloquy [30] can help mitigate any resulting delays. Nonetheless, given the nodes in question are actively processing, though not from disk, our assumption about the NIC being awake is reasonable.

For a period of two week, we traced the system usage of the workstations. Subsequently, we replayed the traces on the systems, measuring the energy consumed by the machines (using Watts up? PRO power meters). Next, we configured the machines to run SARD, and replayed the traces and once again measured the energy. We do not take into account the energy consumed by displays as a simple timeout mechanism will work equally well for them. Also, we factor out long idle periods of system inactivity, e.g., 6PM to 6AM, when SARD has no benefit over a standard energy saving mechanism. This allows us to focus on cases where in the absence of SARD, no energy savings are possible. The total energy consumed by the machines over the duration of the week reduced from 165312 Watt-hour (Wh) to 156895 Wh, a saving of 5.1%. This study shows the potential for SARD to provide energy benefits. Moreover, as long as a copy of an application is available in-memory at some machine in the system, disks at other machines

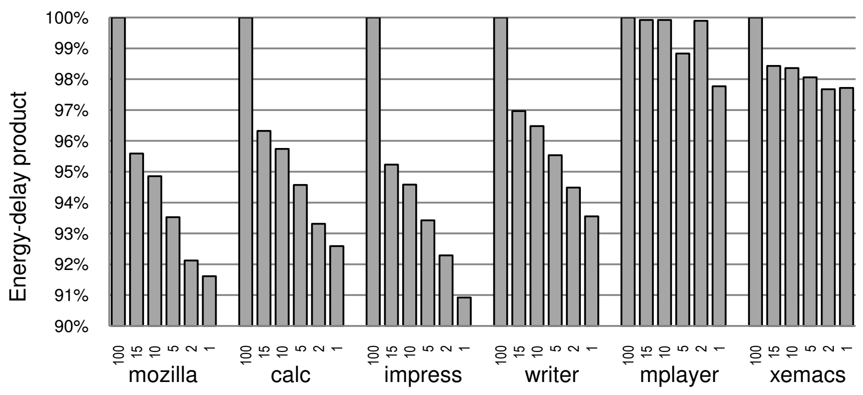

Figure 9. Energy-delay product for the studied applications under various remote servicing conditions, normalized to case of a standalone system (100% local requests).

can stay off. Thus, with a large number of machines as in enterprise environments, SARD has potential to provide even larger savings by keeping more disks in low-power state.

V. Related Work

A. Remote Data Retrieval

We have presented one design approach to supporting remote data-retrieval in SARD. Approaches used in such projects as I/O offloading [31] and p2p-based backup storage systems [32] can also provide mechanisms that can help achieve the remote serving goals of SARD. These works are complementary to SARD, but SARD differs from them in its main focus of reducing energy management delays.

B. Shutdown Prediction Techniques

High performance hard disks are a significant source of power consumption [11]. As a result, numerous timeout based techniques have been proposed to shut down the disk [33]. Moreover, dynamic predictors that shut down the device much earlier than the timeout mechanisms have been investigated [34]. Stochastic modeling techniques have also been applied to model the idle periods in applications and shut down the disk based on the resulting models [35], [36]. Automatic generation of application hints to shut down disks has also been proposed in PCAP [7]. Finally, operating systems can concurrently evaluate multiple predictors and select the best one for the current workload [37].

C. Reducing Spin-up Delays

The goal of shutdown mechanisms is to power down the disk for energy savings. However, every shutdown, even a correct one, will require a corresponding spin-up to serve future requests. There are two approaches to reducing the impact of spin-up delays. Data can be prefetched and cached either in main memory [38] or alternate storage devices such as flash memories [10]. Thus traffic shaping using caching and prefetching can reduce the frequency of spin-ups. By bringing needed data in memory ahead of time and retaining it, associated disk access can be avoided when a disk is shutdown. As a result, fewer spin-up latencies are exposed to the application and the user, and the disk remains shut down longer resulting in higher energy savings. Alternatively, the disk can be actively woken up early by spinning up the platters before the request arrives and serving the request without any delays [9]. Both approaches are complementary to SARD, since the disk will have to be spun up at some point even if the caching techniques are very efficient.

D. Shared Memory Systems

An alternative source of binaries can be obtained by utilizing other computers in the system. The concept of reading binaries/files or using the memories of other machines in a cluster has been discussed in NOW [39]. Given technology trends, remote memory access over the network may be faster than local disk access. SARD share this observation with NOW, however, differ in that SARD does not require striping of data across disks of participant nodes which keeps many disks busy, rather SARD aims to create opportunities for keeping the disks idle. Moreover, in shared memory approaches, it is crucial to minimize the total cost of memory references within a cluster [40] to provide high performance and achieve a low-overhead implementation, unlike SARD.

A simpler approach considers the memory of individual workstations as belonging to a single large pool [41]. In this case, caching of files, especially the read-only text segments of heavily used utilities such as editors and command shells, can be performed in memories of the systems in the network. In addition, dedicated servers with large memories can be used to improve file caching [42]. Alternatively, Cooperative caching [14] relies on a cooperation of machines to provide sharing of the buffer cache. SARD differs from cooperative caching because it does not require participants to have unique data that is necessary for efficiency in cooperative caching. Furthermore, it allows multiple nodes to have copies of the same data, thus avoiding many of the shortcoming due to exclusive data placement that plagues cooperative caches employed in distributed file systems [43] and web proxies [44].

From the above discussion, it is clear that SARD is more than just a simple remote "read" of data over the network. We deploy a discovery mechanism to avoid the need for a dedicated network server to serve binaries. In addition, to the best of our knowledge, such remote retrieval has not been exploited to hide power-savings latency. Thus, SARD is indeed novel from this aspect.

VI. Conclusion

In this paper, we have presented the design and evaluation of SARD, a p2p-based system that mitigates delays associated with disk energy management by allowing sharing of in-memory application images across peers. SARD transparently retrieves application images from nodes on which the applications are already loaded, thus minimizing the need for costly disk accesses and hiding the spin-up delays from the users. Remote application retrieval also provides opportunities for keeping the local disks off, thus saving energy. Our evaluation of SARD shows an average performance improvement of 33.5% for typical desktop applications, and a case study with real usage shows average energy savings of 5.1%. Most importantly, a 71% reduction in delays associated with energy management is observed. Finally, the effect of remote application retrieval on remote nodes is shown to be minimal (less than 3%), and demonstrates that SARD can serve as a practical and effective tool for mitigating energy management delays, which also improves energy efficiency in enterprise environments.

Acknowledgments

We are thankful to the anonymous reviewers and our shepherd Ranveer Chandra for their valuable feedback in preparing the final version of this paper. This research is supported in part by funds from an NSF CAREER grant (CCF-0746832) and IBM Faculty Award (VTF-874574).

References

[1] G. Wang, A. R. Butt, and C. Gniady, "Mitigating disk energy management delays by exploiting peer memory," in *Proc. MASCOTS*, 2009.

[2] R. Bianchini and R. Rajamony, "Power and energy management for server systems," Department of Computer Science, Rutgers University, Tech. Rep. DCS-TR-528, June 2003.

[3] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle, "Managing energy and server resources in hosting centers," in *Proc. SOSP*, 2001.

[4] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy conservation policies for web servers," in *Proc. USITS*, 2003.

[5] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Proc. COLP*, 2001.

[6] Tufts, "Computers and energy efficiency," Online Specification, 2008, http://www.tufts.edu/tie/tci/Computers.html.

[7] C. Gniady, A. R. Butt, Y. C. Hu, and Y.-H. Lu, "Program counter-based prediction techniques for dynamic power management," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 641–658, 2006.

[8] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G. D. Micheli, "Quantitative Comparison of Power Management Algorithms," in *Proc. DATE*, 2000.

[9] I. Crk and C. Gniady, "Context-aware mechanisms for reducing interactive delays of energy management in disks," in *Proc. USENIX ATC*, 2008.

[10] E. B. Nightingale and J. Flinn, "Energy efficiency and storage flexibility in the blue file system," in *Proc. OSDI*, 2004.

[11] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: helping disk arrays sleep through the winter," in *Proc. SOSP*, 2005.

[12] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "Srcmap: Energy proportional storage using dynamic consolidation," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10)*, February 2010.

[13] P. Sehgal, V. Tarasov, and E. Zadok, "Evaluating performance and energy in file system server workloads," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10)*, February 2010.

[14] M. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *Proc. OSDI*, 1994.

[15] M. R. Hines, J. Wang, and K. Gopalan, "Distributed anemone: Transparent low-latency access to remote memory in commodity clusters," in *Proc. HiPC*, 2006.

[16] AMD, "Magic packet technical white paper," 2008, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/20213.pdf.

[17] G. Necula, "Proof carrying code," in *Proc. POPL*, 1997.

[18] J. K. Hollingsworth and E. L. Miller, "Using content-derived names for configuration management," in *SSR '97: Proceedings of the 1997 symposium on Software reusability*. New York, NY, USA: ACM, 1997, pp. 104–109.

[19] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proc. SOSP*, 2001.

[20] A. R. Butt, T. A. Johnson, Y. Zheng, and Y. C. Hu, "Kosha: A peer-to-peer enhancement for the network file system," *Jrnl. of Grid Computing*, vol. 4, no. 3, pp. 323–341, 2006.

[21] A. R. Butt, R. Zhang, and Y. C. Hu, "A self-organizing flock of Condors," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 145–161, 2006.

[22] C. Miller, A. R. Butt, and P. Butler, "On utilization of contributory storage in desktop grids," in *Proc. IPDPS*, 2008.

[23] H. Monti, A. R. Butt, and S. S. Vazhkudai, "Timely offloading of result-data in hpc centers," in *Proc. ACM ICS'08*, 2008.

[24] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM Middleware*, 2001.

[25] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. ACM SOSP*, 2007.

[26] Network Appliance Inc., "PostMark File system Performance Benchmark," Jan 2008, http://www.netapp.com/tech_library/3022.html.

[27] J. Steffen and H.-B. Bröker, "CSCOPE," Jan 2008, http://cscope.sourceforge.net/.

[28] A. R. Butt, C. Gniady, and Y. C. Hu, "The performance impact of kernel prefetching on buffer cache replacement algorithms," *IEEE ToC*, vol. 56, no. 7, pp. 889–908, 2007.

[29] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 9, pp. 1277–1284, Sep 1996.

[30] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy:Augmenting Network Interfaces to Reduce PC Energy Usage," in *Proc. NSDI*, 2009.

[31] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," in *Proc. USENIX FAST*, 2008.

[32] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 285–298, 2002.

[33] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-down Policies for Mobile Computers," in *Proc. USENIX MLICS*, 1995, pp. 381–413.

[34] C.-H. Hwang and A. C. Wu, "A Predictive System Shutdown Method for Energy Saving of Event Driven Computation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 226–241, April 2000.

[35] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, "Policy Optimization for Dynamic Power Management," *IEEE TCAD*, vol. 18, no. 6, pp. 813–833, 1999.

[36] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," in *Proc. DAC*, 1999.

[37] A. Weissel and F. Bellosa, "Self-learning hard disk power management for mobile devices," in *Proc. IWSSPS*, 2006, publication.

[38] J. P. Rybczynski, D. D. E. Long, and A. Amer, "Expecting the unexpected: adaptation for predictive energy conservation," in *Proc. StorageSS*, 2005.

[39] T. Anderson, D. Culler, and D. Patterson, "A case for Networks of Workstations," *IEEE Micro*, vol. 15, no. 1, pp. 54–64, 1995.

[40] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath, "Implementing global memory management in a workstation cluster," in *Proc. SOSP*, 1995.

[41] M. Flouris and E. P. Markatos, "The network ramdisk: Using remote memory on heterogeneous NOWs," *Cluster Computing*, vol. 2, no. 4, pp. 281–293, 1999.

[42] E. P. Markatos and G. Dramitinos, "Implementation of a reliable remote memory pager," in *Proc. USENIX*, 1996.

[43] D. Muntz and P. Honeyman, "Multi-level caching in distributed file systems," in *Proc. Usenix Winter*, 1992.

[44] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Proc. ACM SOSP*, 1999.