QMD: Exploiting Flash for Energy Efficient Disk Arrays

Sean M. Snyder[†] Shimin Chen[⋆] Panos K. Chrysanthis[†] Alexandros Labrinidis[†]
[†]University of Pittsburgh [⋆]Intel Labs

ABSTRACT

Energy consumption for computing devices in general and for data centers in particular is receiving increasingly high attention, both because of the increasing ubiquity of computing and also because of increasing energy prices. In this work, we propose QMD (Quasi Mirrored Disks) that exploit flash as a write buffer to complement RAID systems consisting of hard disks. QMD along with partial on-line mirrors, are a first step towards energy proportionality which is seen as the "holy grail" of energy-efficient system design. QMD exhibits significant energy savings of up 31%, as per our evaluation study using real workloads.

1. INTRODUCTION

A growing concern, energy consumption in data centers has been the focus of numerous white papers, research studies, news reports, and recent NSF workshops [4, 23, 10, 7, 1, 2]. According to a report to U.S. congress [23], the total energy consumption by servers and data centers in U.S. was about 61 billion kWh in 2006, and is projected to nearly double by 2011 [23]. To make matters worse, global electricity prices increased 56% between 2002 and 2006 [7]. The two trends of growing energy consumption and rising energy prices lead to increasingly higher electricity bills for data centers. Energy cost can become a dominant factor in the total cost of ownership of computing infrastructure [4], and the annual electricity cost of data centers in U.S. in 2011 is projected to be $7.4 billion [23]. Among the components in data centers, it has been shown that storage experienced the fastest annual growth (20% between 2000 and 2006) in energy consumption [23]. As hard disk drives (HDDs) are the dominant technology for data storage today, we are interested in improving energy efficiency for data storage that consists of mainly HDDs.

A key goal in energy efficient system design is to achieve energy proportionality [5], i.e., energy consumption being proportional to the system utilization. Unlike solid state devices, such as microprocessors, hard disk drives (HDDs) contain moving components, making it difficult to achieve this goal. For example, for an enterprise class disk, the idle power for spinning the disk platters is often about 60–80% of its active energy [20, 21]. While a disk can be spun down to standby mode for saving most of this power, it takes on the order of 10 seconds to spin up a disk, potentially incurring significant slowdowns in application response times.

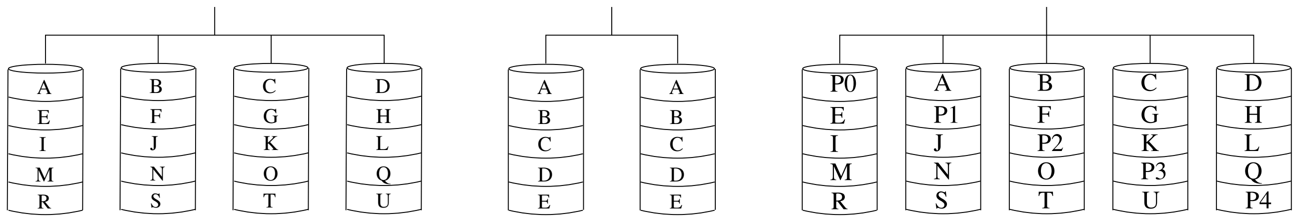
Proceedings of the Seventh International Workshop on Data Management on New Hardware (DaMoN 2011), June 13, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0658-4 ...$10.00.

Previous Approach: Exploit Redundancy and NVRAM. One promising solution is to exploit the inherent redundancy in storage systems for conserving energy [11, 24, 16]. Today, most storage systems employ redundancy (e.g., RAID) to achieve high reliability, high availability, and high performance requirements for many important applications. For example, the TPC-E benchmark, which models transaction processing in a financial brokerage house, requires redundancy for the data and logs [22]. As seen by published TPC-E reports on the TPC web site, this requirement is typically achieved by the use of RAID disk arrays.

For saving energy, the idea is to keep only a single copy of the data active and spin down disks containing redundant copies of the data under low load. We call the disks containing the active copy of data, the *active* disks, and the disks that are spun down, the *standby* disks. Note that the approach must guarantee the same level of reliability for write operations under low load (e.g., writing to two non-volatile devices). To achieve this, previous studies [11, 24, 16] propose to use NVRAM (i.e., battery-backed RAM) as non-volatile write buffers. When the system is under low utilization, reads are sent to the active disks, while writes are sent to both the active disks and to the NVRAM buffers. When the system sees high load or when the NVRAM buffers are full, the standby disks are spun up and the buffered writes are applied. Depending on the RAID organization, this approach may potentially save up to 50% energy when the system is under low utilization.

Limitation of NVRAM-Based Approach. Besides concerns about correlations between battery failure and power loss, battery-backed RAM is expensive. The NVRAM write buffer size is often limited to a few hundred MB for an entire RAID array. Typically, a server-class disk can support about 100MB/s read/write bandwidth. Suppose that under low load, a disk sees 1MB/s write traffic. Then, a 500MB NVRAM buffer will be filled up for the write traffic of a single disk in less than 9 minutes. When the buffer is filled, the standby disks must be spun up to apply the buffered writes. However, a disk supports only a limited number of spin-up/down operations because they introduce wear to the motor and the heads in a disk. In particular, server and desktop disks are often rated at 50,000 spin-up/down cycles (a.k.a. start-stop cycles) [19]. Given a five-year lifetime, this puts a limitation of an average 1.1 spin-up/down per hour. Therefore, the above example with a *single* standby disk will significantly shorten the disk lifetime by about 6 times! Note that in real-world disk arrays, an NVRAM buffer in a RAID controller often handles tens of disks, and thus the situation could be an order of magnitude worse.

Our Solution: Exploiting Flash as Write Buffer. We propose to exploit flash as the write buffer for addressing this problem. There are several desirable properties of flash: (i) it is non-volatile; (ii) flash is much cheaper with much larger capacity; and (iii) flash is



(a) RAID 0 (striping) (b) RAID 1 (mirrored disks) (c) RAID 5 (striping with distributed parity)

Figure 1: Basic RAID schemes. (Capital letters represent data blocks; Px represents a parity block.)

energy efficient and supports energy proportionality well. Moreover, flash-based cache products with hundreds of GB capacity are already available for storage systems [13]. One can utilize the same flash for saving energy. This nicely shares the resource: *the flash-based cache improves I/O performance under high system load and saves energy under low load.*

In our design, we aim to achieve low spin-up/down counts while preserving the performance of RAID under low utilization, under high utilization, and during state transitions. We call the solution in this paper, QMD (Quasi Mirrored Disks), as we mainly focus on mirror-based RAID schemes (e.g., RAID 1 and RAID 10), and we study partial on-line mirrors in order to achieve the ideal goal of energy proportionality. We present preliminary evaluation of our solution by simulating I/O traces of real-world applications. Experimental results show that QMD can save 11%–31% energy, and reduce the number of spin-up/downs by 80%.

Outline. Section 2 provides background on exploiting redundancy for RAID arrays. Section 3 presents our solution, QMD, and discusses our research direction for achieving the energy proportionality goal. Section 4 evaluates QMD using real-world I/O traces. Section 5 describes related work. Section 6 concludes the paper.

2. EXPLOITING STORAGE REDUNDANCY FOR SAVING ENERGY

We start by refreshing our memory of the common RAID schemes in Section 2.1. Then, in Section 2.2, we describe the basic operations for exploiting RAID redundancy for saving energy.

2.1 Background: Common RAID Schemes

Figure 1 shows three basic RAID schemes [14]. They are widely used and serve as building blocks for composite RAID schemes.

- *RAID 0 (a.k.a. striping)* places data blocks across disks in a round robin fashion for high performance. RAID 0 does not provide redundancy, and therefore is often combined with other RAID schemes for reliability and availability.
- *RAID 1 (a.k.a. mirrored disks)* mirrors data blocks between two (or more) disks. Every write is sent to both disks, while a read can be served by either disk. Therefore, RAID 1 of two disks achieves twice the read bandwidth of a single disk and 100% data redundancy. It can tolerate one disk failure.
- *RAID 5 (a.k.a. striping)* stripes data across N ($N \geq 3$) disks. The N blocks at the same disk address form a stripe group. One of them is a parity block, computed as the bit-wise XOR of the other $N-1$ data blocks. RAID 5 places the parity blocks across disks in a round robin fashion. Every write must modify both the target data block and the associated parity block, requiring two reads for fetching the two old blocks followed by two writes. RAID 5 can tolerate a single disk failure. A data block of the failed disk can be reconstructed as the bit-wise XOR of the other $N-1$ blocks in the same stripe group.
- *Composite RAID Schemes:* RAID 10 (i.e., 1+0) is a stripe of mirrors, where every disk in RAID 0 (in Figure 1(a)) is replaced with a pair of mirrored disks. A RAID 10 of $2N$ disks can tolerate N disk failures (each in a disjoint mirror). Similarly, RAID 50 replaces every disk in RAID 0 with a RAID 5.

Redundancy is achieved by either mirror-based schemes (e.g., RAID 10) or parity-based schemes (e.g., RAID 5). The main advantage of the latter is that it saves disk capacity; RAID 5 of N disks utilizes $1 - \frac{1}{N}$ of the total capacity, while RAID 1 utilizes only 50%. In other words, given the same individual disk capacity and the target total capacity, RAID 5 uses fewer number of disks than RAID 10. However, as disk capacity has been growing exponentially, total capacity is less of a concern today. The number of disks in RAID is often determined by application performance requirements in terms of throughput and IOs per seconds (IOPS), rather than total available capacity, as evidenced by many TPC results.

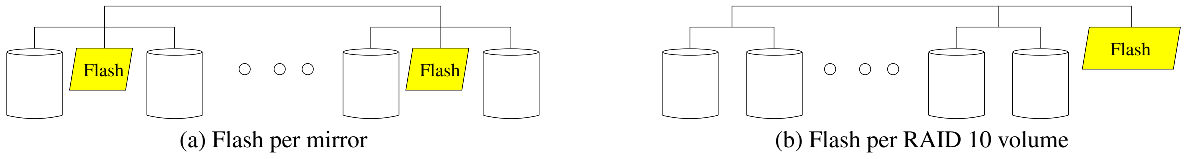
On the other hand, mirror-based schemes have higher write performance than parity-based schemes during normal operations. Moreover, when a disk fails, mirror-based schemes can serve data directly from good disks, while parity-based schemes suffer from large performance degradation due to the many I/Os for retrieving blocks in the same stripe group to reconstruct a block on the failed disk. As a result, mirror-based schemes become increasingly popular today. For example, many TPC-E benchmark results that store data on disks use RAID 10 for reliability.[1] We focus on mirror-based schemes in this paper.

2.2 Basic Operations for Saving Energy

Previous studies [11, 24, 16] propose to exploit the RAID redundancy for saving energy and uses NVRAM for achieving reliability when the system is under low utilization. The basic operations are:

1. *All Disks under High Utilization:* The system performs normal RAID operations with all disks running under high load.
2. *Active Disks with NVRAM under Low Utilization:* When the system is under low utilization, a number of disks is spun down to save energy. This number depends on the RAID scheme. In mirror-based schemes, one disk in every mirror can be spun down, thus potentially saving up to 50% energy. In parity-based schemes (e.g., RAID 5 with N disks), one disk in every parity group can be spun down (saving up to $\frac{1}{N}$ energy in RAID 5). Reads are handled by the active disks; parity-based schemes require the costly XOR computation for reconstructing disk blocks on the standby disks. Writes are sent to both

[1]A few published TPC-E results (including the current Watts/Performance lead, Fujitsu PRIMERGY RX300 S6 12x2.5) store data mainly on arrays of flash-based SSDs for reducing energy consumption. In such configurations, SSD capacity is a much more significant concern and therefore RAID 5 is often employed. However, the focus of this paper is on using a small amount of flash for improving the energy efficiency of disk storage, which is the dominant storage technology today.



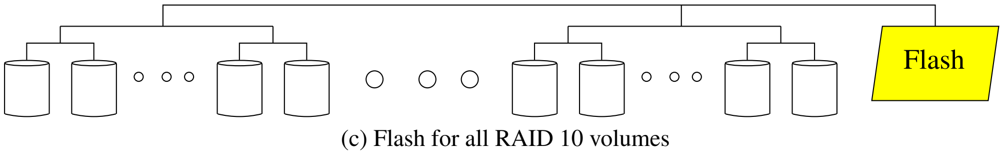


Figure 2: Flash-enhanced RAID 10 schemes.

the active disks and the NVRAM to guarantee reliability. Comparing mirror-based and parity-based schemes, it is clear that the approach fits mirror-based schemes better.

3. *Flushing NVRAM Data to Standby Disks:* When the NVRAM write buffer is full or when the system transitions from low utilization back to high utilization, the data cached on NVRAM must be flushed to bring the standby disks up to date.

There are two main problems of this approach. First, the capacity of NVRAM is typically limited to a few hundred MB, potentially incurring frequent flushing operations, as shown in the back-of-envelope computation in the Introduction. Frequent flushing operations can both dramatically reduce disk lifetimes and reduce the energy savings because the standby disks must be spun up for writing the buffered data. In this paper, we address this problem by exploiting flash as a large-capacity nonvolatile write buffer.

Second, the energy savings are bounded by the RAID schemes, leaving a big gap to reach the ideal goal of energy proportionality. For example, when the system is under 1% load, RAID 10 still keeps 50% of the disks active. We discuss potential solutions to this problem, which requires coordination between applications and storage systems to avoid performance problems.

3. QUASI MIRRORED DISKS

We propose QMD (Quasi Mirrored Disks) that exploits flash for removing the limitation of NVRAM in Section 3.1, and discuss *partial on-line mirrors* as future research direction for achieving the goal of energy proportionality in Section 3.2.

3.1 Flash-Based Write Cache

We analyze the access pattern of the write cache. Under low utilization, block writes are appended to the write cache, resulting in sequential writes. During flushing, the write cache must support random reads for two reasons: (i) we would like to reorder and optimize the block writes to disks; and (ii) during the transition from low utilization to high utilization mode, we want the write cache to serve incoming I/O requests in order to minimize the impact of flushing on front-end operations. As flash supports both the above access patterns well, we propose to use flash as the write cache.

Figure 2 shows three ways to include flash-based write caches in QMD, using RAID 10 as an example RAID scheme. In (a), we enhance every pair of mirrored disks with a separate flash device. In (b), we use a flash device for an entire RAID 10 volume that stripes data across the mirror pairs. In (c), multiple RAID 10 volumes share the same flash device. From (a) to (c), we reduce the number of flash devices in the system. The benefits are two folds: lowering the total cost and allowing dynamic sharing of the flash capacity across disks. The latter is especially important for multiple RAID 10 volumes because different volumes present separate

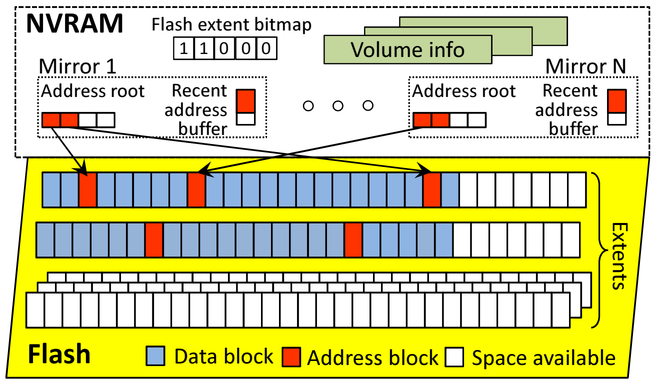


Figure 3: QMD data structures.

I/O address spaces to software and typically store different files. Therefore, they may see very different utilizations. For example, some volumes may be under high utilization, while others under low utilization. The volumes under low utilizations may see different write traffic and thus consume the write cache at different speeds. Dynamic sharing can balance the spin-up/down (flushing) frequencies across different volumes and achieve the same maximum flushing frequency with reduced total write cache size.

However, dynamic sharing introduces complexities in flash cache management. For example, a naive design may employ a single log-structured layout for the flash space. However, different volumes may perform the flush operations at different times, leaving many holes in the log. This either incurs random writes or requires expensive garbage collection operations. In the following, we describe our proposal to efficiently address these complexities.

Flash Space Management. Figure 3 illustrates the data structures of QMD on flash and in NVRAM. Since most functions of NVRAM (e.g., non-volatile write cache during both high and low utilizations) can be satisfied by the flash cache, cost-effective storage designs may reduce the size of NVRAM significantly (e.g., by 10X). Therefore, we use only a small amount of NVRAM in our design mainly as a staging area to reduce wasteful flash writes.

The flash space is divided into fixed sized (e.g., 1GB) extents, as shown in Figure 3. Extent is the unit of flash space allocation. A bitmap in NVRAM keeps track of the availability of all extents. A RAID 10 volume allocates a flash extent at a time. Writes to the volume are appended to the current extent; only when the extent is filled does the volume allocates a new extent. Let V be the number of volumes, C the flash capacity, and E the extent size. We choose E to satisfy $\frac{C}{E} >> V$, i.e., the number of extents is much larger

```
1: Load address blocks of this mirror into memory and sort the address
   entries in disk address order (let $A[0..M-1]$ be the sorted array, where
   $M$ is the total number of entries);
2: $Queue$ is the incoming request queue for this mirror;
3: $ToWrite = M$; $Last = -1$; $Direction = 1$; $Bound = M$;
4: while $ToWrite > 0$ do
5:   while $Queue.not\_empty$ && $Queue.head$ is a write do
6:     $R = Queue.dequeue()$;
7:     Record $R$'s data in flash and $R$'s target address in NVRAM;
8:   end while
9:   if $Queue.not\_empty$ && $Queue.head$ is a read then
10:    $R = Queue.dequeue()$;
11:    Search $R$'s disk address in $A[...]$, using binary search;
12:    if there is a match then
13:      Complete request $R$ by reading the data block from flash;
14:    else
15:      Send $R$ to the disk;
16:      $A[i]$ is immediately to the left of $R$;
17:      if $Last < i$ then
18:        $Direction = 1$; $Bound = M$; $Last = i$;
19:      else
20:        $Direction = -1$; $Bound = -1$; $Last = i + 1$;
21:      end if
22:    end if
23:  end if
24:  $WriteBack = 0$;
25:  for ($j = Last + Direction$; ($j\ != Bound$) &&
       ($WriteBack < K$); $j = j + Direction$) do
26:    if $A[j]$ is valid then
27:      Process $A[j]$: read its data block from flash and send write
         request to the disk;
28:      Mark $A[j]$ to be invalid;
29:      $WriteBack++$;
30:    end if
31:  end for
32:  $ToWrite = ToWrite - WriteBack$; $Last = j$;
33: end while
```

Figure 4: Flush operation for a pair of mirrored disks.

than the number of volumes. In this way, the flash space can be shared to effectively balance the needs of multiple volumes[2].

For every volume, we keep a volume info structure in NVRAM. This structure records all the extents that belong to the volume, the next flash offset to write, and a 1MB sized staging area for flash writes. The latter ensures that writes are performed in large sizes, thereby avoiding the problems of small random flash writes.

Handling Writes under Low Utilization. A write request consists of a data block and the target disk address. We append the data block to the current flash extent, and generate an address entry: (target disk address, flash address of data). However, it is wasteful to incur a (e.g., 4KB) flash write for the small sized address entry. Instead, we store it in a recent address buffer in NVRAM. When this buffer is full, we flush the buffer as an address block to the current flash extent.

As shown in Figure 3, we use a two-level structure to keep track of the data blocks. To facilitate the flushing operation for every pair of mirrored disks, we maintain this structure for every mirror. The first level is the address root in NVRAM, which records the flash offsets of the address blocks for the mirror. The second level consists of the address blocks, which in turn point to the data blocks in flash extents.

Optimizing the Flushing Operation. The system decides to transition a RAID volume from low utilization to high utilization mode

[2]For example, if $C = 100GB$ and $V = 10$, we can choose $E = 1GB$. Then, the scheme can effectively handle even extreme cases such as one volume seeing 90X write traffic than the other volumes.

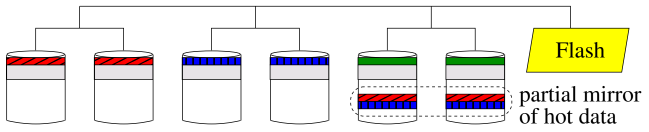


Figure 5: QMD partial on-line mirror. (Red/blue/green: address range for hot data; gray: address range for cold data)

by monitoring the I/O activities. (The implementation will be described in Section 4.) It spins up the standby disks and applies the buffered writes to bring the disks up to date. We would like to optimize the flushing operation for two goals: (i) efficiently writing the buffered data blocks; and (ii) servicing incoming requests during the transition. The key to achieve these goals is to reorder the disk writes to avoid disk seeks.

Figure 4 shows the algorithm for the flushing operation. We can run this algorithm for every pair of mirrored disks in parallel. The algorithm assumes that the volatile DRAM in the storage controller is large enough to hold all the address entries. At the beginning of the algorithm (Line 1), it reads all the address blocks of the mirror and sort the address entries. Sorting serves two purposes: (i) we can easily search incoming read requests to see if there is a hit (Line 11–13); (ii) schedule the write-backs in a disk friendly manner. The algorithm goes into a loop (Line 4–33), which processes K write backs in every iteration. (K is an algorithm parameter that we determine experimentally in Section 4.2.) An iteration first checks incoming requests. For incoming write requests, we simply cache them in the flash and NVRAM (Line 5–8). If there is an incoming read request and it is not a hit, then we send the request to the disk (Line 15). The algorithm remembers the last disk head position, and the previous direction of head movement. It chooses to schedule K write backs from the last head position following the same head movement (Line 25–31).

Space Requirement. In NVRAM, we keep a 1MB sized buffer per RAID volume, and a 4KB recent address buffer per mirror. Suppose there are 10 volumes, and 1000 mirrored disk pairs. Then we require 14MB NVRAM space, which is quite modest.

We also require volatile DRAM for the flushing operation. An address entry consists of a disk block address, and an offset in the flash extent. We can use a 32-bit integer for both addresses. For 4KB blocks, this is sufficient to support 16TB capacity. Therefore, an address entry takes 8 bytes, a 1:512 size ratio to the block size. Suppose the flash cache is 512GB large, then we require 1GB DRAM for the flushing operation, which is reasonable for a large disk array system.

3.2 Partial On-line Mirrors

By exploiting RAID redundancy, we can save at most 50% energy in mirror-based RAID schemes. (The savings in parity based schemes are even smaller.) There is still a big gap to the energy proportionality goal. In this subsection, we discuss our ideas for closing this gap that we would like to explore in future research.

To further save energy, we have to spin down more disks. Therefore, not all data can be available on the active disks. While storage systems may guess the future access patterns based on block-level access history, the penalty of wrong guesses (i.e., the spin-up delay) is high especially for latency sensitive applications.

We propose to allow application software (e.g., database system) and storage systems to collaborate on addressing this problem. Compared to a storage-only solution, application software has higher-level knowledge about data accesses, has more flexibility to schedule data accesses, and can also make end users aware of the relationship of energy consumption, performance, and data placement. We propose the following two interfaces between application software and storage systems:

- Software can divide the address range of a RAID volume into hot and cold address ranges, as shown in Figure 5. For example, DBMS can create a hot and a cold table spaces. It determines the temperature of database objects based on high-level knowledge of user workloads, then stores them in corresponding table spaces. DBMS may opt to expose the choices to the end users (e.g., DBMS admin) showing also the estimate energy costs and response times for query workloads. The storage system guarantees that data in the hot address range will always be available on active disks, while it may take a spin-up delay to access the cold data.
- Software can use an interface to query the status of a cold address range, i.e., whether a spin-up will be necessary to access it. Software may use this information to intelligently schedule its work. For example, if DBMS finds that a database query must access both hot and cold data, DBMS can choose to process the part of the query involving hot data first, and postpone accessing the cold data to hide the spin-up delay as much as possible.

Given the above collaboration, we can spin down disks based on the system utilization. If the system is utilized 50% or more, then we can exploit redundancy as described previously to spin down at most one disk per pair of mirror. If the system utilization is below 50%, we will spin down both disks in some mirrors. However, we must kept the hot data available on active disks.

As shown in Figure 5, we take advantage of the fact that disk capacity is often under utilized. In many important applications, such as OLTP, the number of disks is determined mainly by the performance requirement rather than capacity demands. Therefore, we can copy the hot data to the active mirror, essentially creating a mirrored copy of the hot data. In Figure 5, we plan to spin down the first and second pairs of mirrored disks, and keep the third pair active. Therefore, we copy their hot data to the third pairs of mirrored disks. We call this approach *partial on-line mirrors*.

This approach utilizes all disks under peak utilization, and is capable of spinning down almost all disks for saving energy for low utilization. One major cost is the overhead for copying the hot data. The copying may be improved in two ways. First, we can copy hot data and spin down the mirrors one mirror at a time. For example, in Figure 5, we can copy hot data from the first mirror then spin down it, before copying hot data from the second mirror. This saves energy during the copying process. Second, we may keep an old version of the hot data on the third mirror. Then the copying process needs to only update the old version, potentially avoiding significant fractions of data copying.

4. EXPERIMENTAL EVALUATION

In this section, we present preliminary evaluation of our proposed QMD solution. In Section 4.1, we perform trace-based simulation study using real-world disk traces for quantifying the overall benefits of our solution in terms of energy savings, reduced spin-up/down cycles, and impact on I/O performance. In Section 4.2, we perform real-machine experiments for understanding the benefits of the proposed flushing operation.

4.1 Simulation Study

We implemented a trace driven RAID controller simulator to evaluate the effectiveness of QMD. We used three real workload traces on RAID10 systems and find that (i) significant energy savings can be achieved with minimal impact on response times, and (ii) increasing the non-volatile write buffer size can significantly reduce the number of disk spin down cycles during a trace.

Table 1: QMD default simulator parameters.

Parameter	Default value	Parameter	Default value
Disk block size	512B	Flash read latency	65 us
RAID Stripe size	128KB	Flash write latency	85 us
Power, under load	13.2W	Flash read bandwidth	250MB/s
Power, spinning idle	7.7W	Flash write bandwidth	70MB/s
Power, spun down	2.5W	Epoch length	1 sec
Disk avg. seek time	3.5 ms	Spin down utilization thld.	0.10
Disk avg rot. latency	2.0 ms	Spin down time threshold	30 epochs
Disk transfer rate	120MB/s	Spin up utilization threshold	0.25
Spin up time	10.9 sec	Spin up time threshold	2 epochs
Spin down time	1.5 sec	Flash buffer size	16GB

Simulator Implementation. Our simulator uses block level I/O traces for its workloads. Traces must have four basic fields for each request: arrival time at the controller, read or write, start address, and size. For each request, the simulator progresses time up until the arrival time of the request, then maps the request or pieces of the request to appropriate disks and/or flash based on the current state of the system. When time has progressed to the point that a request at a disk (or flash) finishes, the controller is notified, and response time is taken to be the finish time of the last piece of a request minus the request's arrival time.

The default parameters for the simulator are shown in Table 1. The disk parameters are taken either from values measured in [16] or from the Hitachi Ultrastar 15K600 300GB enterprise drive spec sheet [8]. The flash parameters are taken from the Intel X-25M spec sheet [9].

Disks are not modeled in detail; access time is estimated as average rotational latency plus average seek time plus data transfer time. Sequential accesses are accounted for by removing rotational latency and seek time for accesses following the initial one. Flash access time is similarly modeled as flash read/write latency plus read/write transfer time, with latency removed for sequential accesses. Controller processing time is not accounted for, but is assumed to be insignificant.

Disk utilization is tracked in terms of epochs. Utilization is defined as the amount of time during an epoch a disk spent servicing requests, divided by the total length of the epoch. The threshold for a mirrored pair to transition to or from the low utilization state is set in terms of a utilization and a number of epochs. At the end of each epoch, the state of the system is evaluated to see if any disks should be transitioned to a different state. To transition to the low utilization state, a pair must be below the utilization threshold for the set number of epochs. A pair transitions to high utilization mode either when the utilization of the active disk exceeds its threshold for the set number of epochs, or the space used in the write buffer is above the buffer fill threshold.

Energy used by a hard disk during a trace is computed from the time spent in each of three states times the power used in each state. The states are spinning and serving requests, spinning but idle, and spun down. The energy used by the flash buffer is computed as the time spent idle and active times the power used in each state by the Intel X-25M. Energy savings for the whole system during a trace is computed as the percent difference in energy used when compared to a simulation with no energy savings enabled. The costs due to thermal power (i.e., cooling) are not considered but it is expected that spinning down disks will lead to additional energy savings because of reduced demand for running fans.

Table 2: QMD overall energy savings potential.

Trace file	Description	Length	Peak IO Rate	Energy Savings
cambridge-src1_1	enterprise server	7 days	3.5k/s	31%
cambridge-usr2	enterprise server	7 days	2.3k/s	28%
LiveMaps	global web service	1 day	4.7k/s	11%

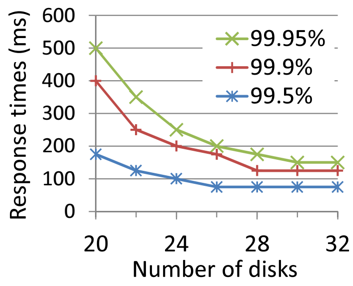


Figure 6: Impact of number of disks on response time (cambridge-src1_1)

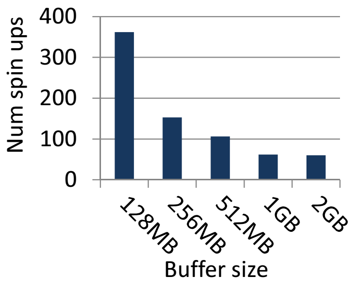


Figure 7: Number of spin ups over 7 days varying buffer size (cambridge-src1_1)

Real-World Traces. We use three real-world traces in our evaluation, as described in Table 2. The first two were recorded on servers in Microsoft Research Cambridge's enterprise data center, and we refer to them as cambridge-src1_1 and cambridge-usr2. The two traces were taken over seven days. They exhibit clear diurnal usage patterns, with peaks during the day and long low utilization periods between them. (The two traces are described in more detail in [12].) The third trace was taken from production Microsoft LiveMaps backend servers over one day. As LiveMaps is a global web service, the third trace shows no diurnal pattern and has very spiky activity throughout the day with only very short low utilization periods.

To evaluate QMD on a given trace, we first perform a range of simulations varying the number of disks without QMD enabled to determine the appropriate number of disks to use for this trace. The number of disks is chosen as the point at which increasing the number further provides minimal response time improvements. This is illustrated in Figure 6. The 99.5th percentile, 99.9th percentile, and 99.95th percentile response times are shown as the number of disks increases for the trace cambridge-src1_1. For this trace, we chose to use 28 disks for further experiments. We also use the simulation without QMD enabled to obtain a baseline energy usage which we use to determine the percent savings when QMD is enabled.

Overall Results. Our overall energy savings results for each trace are shown in Table 2. The energy savings ranges from about 31% in the best case to about 11% in the worst case. The cambridge-src1_1 trace benefits the most from QMD due to its strong diurnal usage pattern, and the cambridge-usr2 trace similarly has very long periods of low utilization. The LiveMaps server's trace has very short low utilization periods, and therefore it is much more difficult for QMD to be effective.

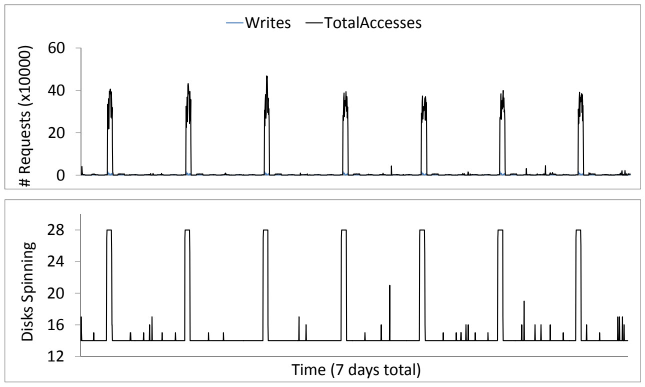


Figure 8: Cambridge-src1_1 - Total IO and writes above, average number of disks spinning below.

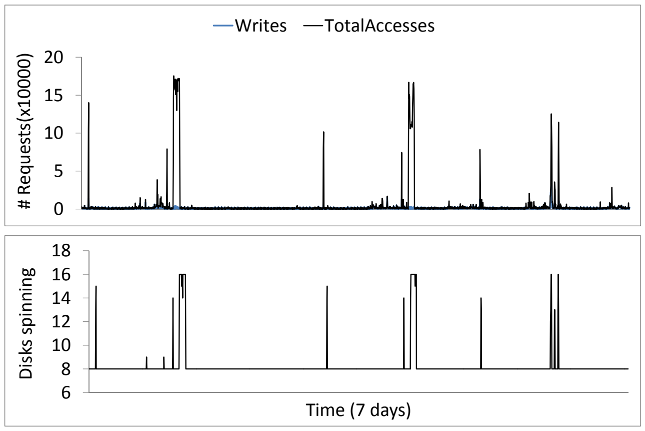


Figure 9: cambridge-usr2 - Total IO and writes above, average number of disks spinning below.

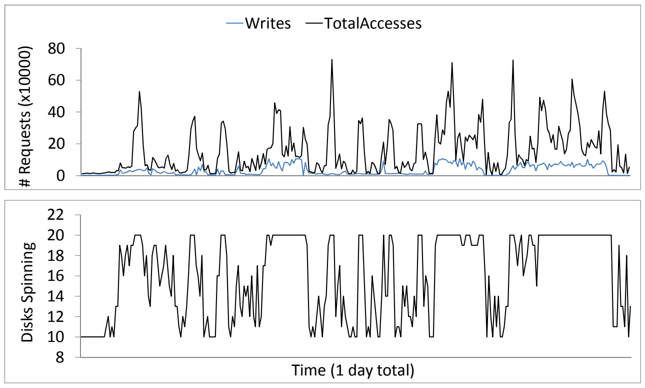


Figure 10: LiveMaps - Total IO and writes above, average number of disks spinning below.

Figure 8, 9, and 10 compare I/O request arrival rates and the average number of active disks over the duration of the traces for the three traces, cambridge-src1_1, cambridge-usr2, and LiveMaps, respectively. The number of disks spinning ranges from 14 (half) during low utilization to 28 (all) during the peak utilization's. From the figures, we see that the spikes of the number of active disks correspond to the spikes in I/O arrival rates in Figure 8 and 9, indicating

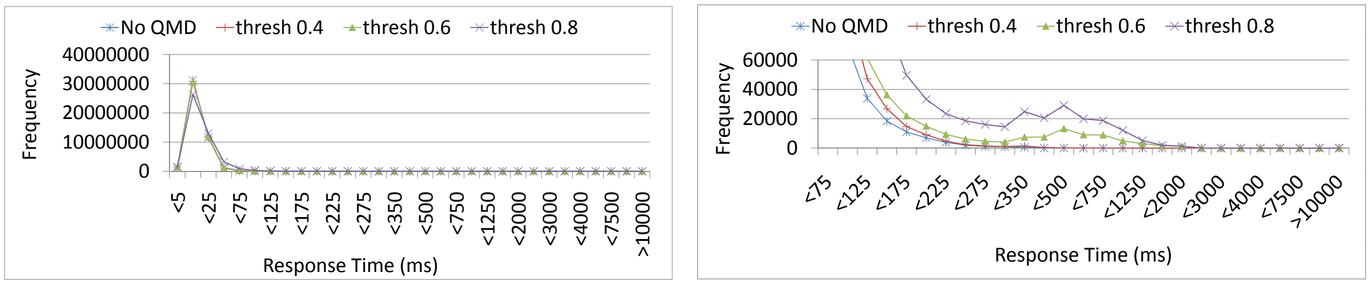


Figure 11: Cambridge-src1_1 - Overall response time distribution left, close up of tail end of distribution right, both shown while varying spin up utilization threshold.

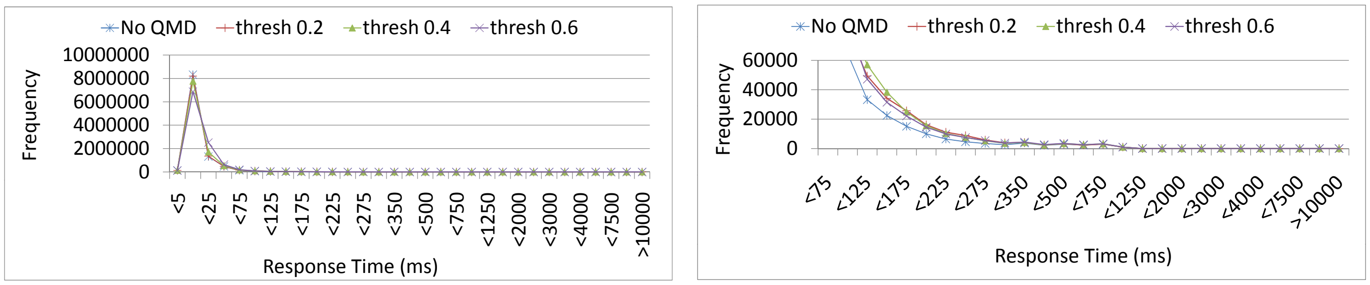


Figure 12: Cambridge-usr2 - Overall response time distribution left, close up of tail end of distribution right, both shown while varying spin up utilization threshold.

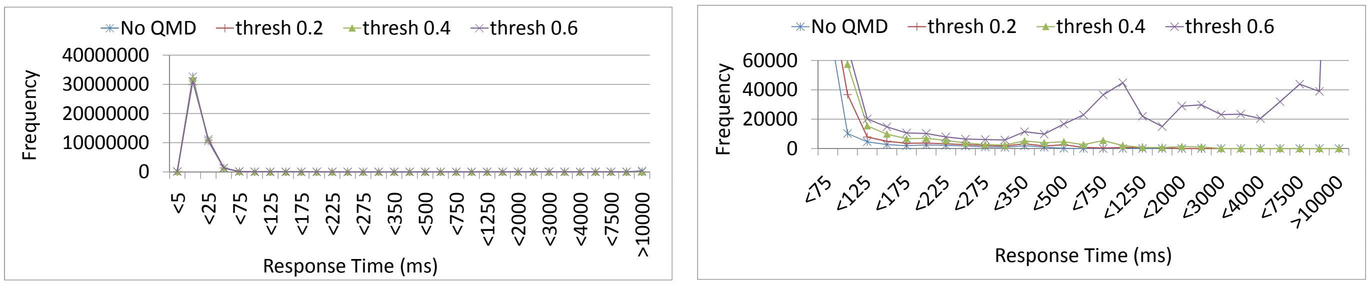


Figure 13: LiveMaps - Overall response time distribution left, close up of tail end of distribution right, both shown while varying spin up utilization threshold.

the effectiveness of QMD for the long periods of low utilization. In contrast, the LiveMaps trace sees bursty traffic with very short durations of low utilization periods, making it challenging to maintain response times and save energy at the same time.

Impact of Write Buffer Size on Spin-Up Cycles. Figure 7 shows the benefit, in terms of disk spin down/up cycles, as the buffer size increases for the trace cambridge-src1_1. epochs. At 128MB buffer size, each disk is spun down an average of 360 times over the seven day trace. As the buffer size is increased to 2GB the number of spin down cycles is reduced to 60 for 7 days, about an 83% reduction. This meets our goal for a five year HDD lifespan.

Impact of QMD on I/O Response Times. Figures 11, 12, and 13 show the effect of QMD on the response time distributions for all three traces. Each figure shows histograms of response time distributions. The horizontal axis is the upper end of response time histogram buckets, and the vertical axis is the number of requests in each bucket. Each figure shows the response time distribution for the original case where QMD is not enabled, and three curves for QMD while varying the spin up utilization threshold.

The left graphs show the overall response time distributions. We see that there is almost no variation at this scale. This means that QMD has little impact for the majority of I/O requests. The right graphs examine more closely the tail end of the distributions. We see that if the spin up utilization threshold is set too high, the scheme can significantly increase the number of requests with very high response times for some workloads (e.g. threshold 0.6 in Figure 13). When the spin up utilization threshold is too high, disks wait too long to start spinning up as I/O arrival rate increases. The active disks get overloaded, and there are deep queues by the time the standby disks have been spun up. However, it is also clear that we can conservatively choose the spin up utilization threshold (which we did for Table 2) so that the response time distribution with QMD enabled almost perfectly follows the distribution with no QMD.

4.2 Opportunity Study for Flushing Operation on Real Machine

Figure 14 shows the real machine experimental results on serving incoming requests while flushing buffered data blocks using the flushing algorithm in Figure 4. We ran the experiments on a Dell PowerEdge 1900 server equipped with two Quad-core 2.0GHz Intel Xeon E5335 CPUs and 8GB memory running 64-bit Ubuntu 9.04

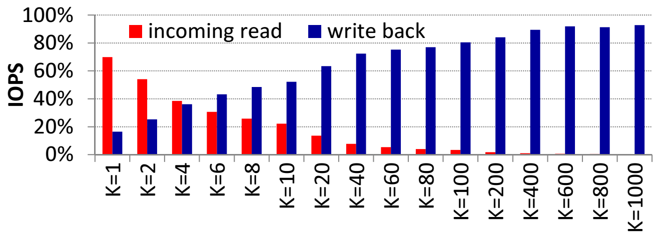


Figure 14: Opportunity study for serving incoming reads while flushing buffered writes as in the flushing algorithm in Figure 4.

server with Linux 2.6.28-17 kernel. We used a dedicated Seagate Barracuda ES.2 [21] enterprise-class drive (750GB, 7200 rpm) as the target disk. To model the buffered data blocks, we randomly generate 100,000 target addresses in the entire disk capacity range. We modeled the worst case for incoming requests: Every iteration in the algorithm processes an incoming read request. The read target addresses are randomly generated. All disk reads and writes are 8KB sized. Every experiment performed all the 100,000 writes.

We vary the number of write backs per iteration (i.e. parameter K) on the X axis in Figure 14. On the Y axis, we report IOs per second normalized to the read-only and write-back-only performance, respectively. We see that as K increases, write-back performance increases while read performance decreases gracefully. For example, at $K = 2$, the algorithm achieve 54% of the peak read and 25% of the peak write performance. At $K = 10$, the algorithm achieve 25% of the peak read, and 48% of the peak write performance. Our algorithm schedules write backs whose target addresses are close to every read request for reducing disk seek overhead. Using the measurements, a storage system can choose a K that balances the write-backs and incoming request handling for given targets of system loads and write-back times.

5. RELATED WORK

Exploiting Redundancy for Saving Disk Energy. As described in Section 1, several previous studies proposed to exploit redundancy in data storage systems to save energy. EERAID [11] and RIMAC [24] exploited redundancy in RAID arrays for saving energy. They focused on request scheduling and storage cache designs. Diverted Access[16] exploited redundancy in a more general form, where storage systems store (encoded) data in n fragments, and every subset of m fragments can be used to reconstruct the data. However, these previous proposals suffer from two problems: (i) Limited NVRAM capacity forces disks to be frequently spun up/down, which impacts their lifetimes; and (ii) there is still a significant gap to achieve the ideal goal of energy proportionality. In this paper, we exploited flash as a larger non-volatile write cache to address (i). For (ii), we proposed partial online mirror and a collaboration interface between storage systems and upper-level software for spinning down more disks while limiting performance impacts.

Migrating or Copying Data for Saving Disk Energy. MAID [6] maintains a large number of disks in standby mode for archival purpose. A disk is spun up on demand for serving a request. To reduce the accesses to standby disks, MAID uses a small number of disks to cache recently used data. PDC [15] migrates frequently used data to a subset of all the disks so that other disks can be spun down. In this paper, we exploit both redundancy and data migration for achieving energy proportionality. In addition, we propose to expose energy state information to upper-level software (e.g., database systems) so that they can collaborate to hide the spin-up delays for accessing the cold data.

Exploiting Flash as Write Buffers. Schindler et al. proposed to use flash as a write cache to optimize storage access patterns that consist of large sequential reads and random writes [17]. Chen et al. exploited a flash-based cache for enabling online updates in data warehouses [3]. While both studies maintain the efficiency of sequential reads in face of random writes, their focuses are quite different. Chen et al. developed a MaSM algorithm, for supporting fine-grain database record updates, minimizing memory footprint and flash writes, and supporting ACID properties [3]. On the other hand, storage systems see only block-sized I/Os and do not require ACID, which simplify the flash management in Schindler et al.'s solution. Instead, they focus on efficiently migrating the cached writes back to disks in the middle of large sequential read operations [17]. In this paper, we also exploit flash as a write cache, but for a very different purpose: significantly increasing the size of the non-volatile write cache for reducing the number of spin-up/down cycles for disks in energy-efficient disk arrays.

Efficient Disk Access Patterns. Sequential accesses and random accesses are the two access patterns that are studied most for disks. Because of their mechanical properties, HDDs achieve peak bandwidth for sequential access patterns but have poor performance for random accesses. In between these two extremes, previous work studied other efficient access patterns for modern disks. Schlosser et al. exploited the semi-sequential pattern and short seeks for multidimensional data layout on disks [18]. On modern disks, short seeks up to a certain number of disk tracks take similar time. A list of disk blocks on different disk tracks satisfies the semi-sequential pattern if the next block on the list can be accessed without rotational delay after seeking from the previous block. Combining these two features, Schlosser et al. identified that from any given disk blocks, there is a set of disk blocks, called adjacent blocks, that can be accessed with equally small cost. Then, they placed multidimensional data using adjacent blocks for efficient accesses in every dimension. Schindler et al. studied proximal I/Os for combining random writes into large sequential reads [17]. Proximal I/Os are a set of I/O requests with addresses close to one another. Modern disks can handle these I/Os with minimal seek overhead. Similar to both of the studies, we also aim to reduce disk seeks by scheduling I/Os with close addresses, but for a quite different workload: flushing a large number of buffered writes to disks while serving (random) incoming requests. Our proposal balances the two activities, and supports performance tuning based on a simple parameter, the number of write backs performed between two incoming requests.

6. CONCLUSION

In this paper, we investigated energy efficiency for HDD-based data storage in general and RAID systems in particular. We proposed QMD (Quasi Mirrored Disks) to effectively leverage redundancy in storage systems and flash to save significant energy. We demonstrated this through simulation using real-world workloads, including systems that experience long periods of low utilization, i.e. due to diurnal usage patterns. With a sufficiently large non-volatile write buffer, the number of spin down cycles for disks can be kept within the average lifetime limit specified by manufacturers. Moreover, we can choose conservative parameters (e.g., spin-up utilization threshold) so that QMD can achieve the energy savings with negligible impact on I/O response times.

We find that exploiting redundancy alone still leaves a big gap to the energy proportionality goal. As future research, we propose two interfaces that allow applications and storage systems to collaborate for further saving energy, and Partial On-line Mirrors for effectively taking advantage of the interfaces.

7. ACKNOWLEDGMENTS

The QMD simulator was initially developed by Katlyn Daniluk as part of her research experience for undergraduates. This work was partially supported by National Science Foundation awards IIS-0746696 and IIS-1050301.

8. REFERENCES

[1] NSF workshop on sustainable energy efficient data management (SEEDM). http://seedm.org.

[2] NSF workshop on the science of power management. http://www.cs.pitt.edu/ kirk/SciPM2.

[3] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. MaSM: Efficient online updates in data warehouses. In *SIGMOD*, 2011.

[4] L. A. Barroso. The price of performance: An economic case for chip multiprocessing. *ACM Queue*, pages 48–53, Sept 2005.

[5] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, 40:33–37, Dec 2007.

[6] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *SC*, 2002.

[7] Emerson Network Power. Energy logic: Reducing data center energy consumption by creating savings that cascade across systems. http://emersonnetworkpower.com/en-US/Brands/Liebert/Documents/ White Papers/Energy Logic_Reducing Data Center Energy Consumption by Creating Savings that Cascade Across Systems.pdf.

[8] Hitachi Global Storage Technologies. Ultrastar 15k600 data sheet, Sept 2009.

[9] Intel Corporation. X-25M SATA SSD 34nm product specification.

[10] J. G. Koomey. Estimating total power consumption by servers in the U.S. and the world. http://enterprise.amd.com/Downloads/ svrpwrusecompletefinal.pdf.

[11] D. Li and J. Wang. EERAID: energy efficient redundant and inexpensive disk array. In *ACM SIGOPS European Workshop*, 2004.

[12] D. Narayanan, A. Donnelly, and A. I. T. Rowstron. Write off-loading: Practical power management for enterprise storage. In *FAST*, 2008.

[13] NetApp Corporation. Flash cache. http://www.netapp.com/us/ products/storage-systems/flash-cache/flash-cache.html.

[14] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *SIGMOD*, 1988.

[15] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *ICS*, 2004.

[16] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. In *SIGMETRICS*, 2006.

[17] J. Schindler, S. Shete, and K. A. Smith. Improving throughput for small disk requests with proximal I/O. *FAST*, 2011.

[18] S. W. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. R. Ganger. On multidimensional data and modern disks. In *FAST*, 2005.

[19] Seagate Technology LLC. Barracuda 7200.12 data sheet.

[20] Seagate Technology LLC. Cheetah 15K.4 SCSI product manual, rev. d edition, May 2005. Publication number: 100220456.

[21] Seagate Technology LLC. Barracuda ES.2 data sheet, 2009.

[22] Transaction Processing Performance Council. TPC benchmark E standard specification version 1.12.0.

[23] US Environmental Protection Agency. Report to congress on server and data center energy efficiency: Public law 109-431.

[24] X. Yao and J. Wang. RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. In *EuroSys*, 2006.