Caching according to use and QoS requirements

Etai Hazan
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel 8410501
etai@cs.bgu.ac.il

Mark Abashkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel 8410501
abashkin@cs.bgu.ac.il

Gal Lipetz
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel 8410501
lipetzg@cs.bgu.ac.il

Assaf Natanzon
EMC Corp., Tel Aviv, Israel and Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel 8410501
assaf.natanzon@emc.com

Eitan Bachmat
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel 8410501
ebachmat@cs.bgu.ac.il

Abstract

In this paper we consider the management of a two layered cache, consisting of a DRAM cache and a secondary flash based cache, with the challenge of taking into account heterogeneous quality of service (QoS) requirements.

We introduce a flexible method for controlling the level of cache usage by different data extents according to their user defined priority and their ability to take advantage of cache as a resource.

We show that our method without taking priorities into account improves the cache hit ratio. When we take priorities into account, the response times of the high priority data improves, while the overall hit ratio still improves, thus our framework achieves both purposes.

Our experiments use trace data from real enterprise production systems.

We then show that our method can be combined with more classical optimization methods and still lead to improvements.

1. Introduction

An important challenge which is facing enterprise storage systems, especially in virtualized and cloud environments is Quality of Service (QoS). In many cases quality of service is defined in terms of some performance measures, such as average response time. The user requires that various portions of their data set will achieve various performance measurement targets. As an example, the user may state that data set A requires average response time of $2ms$, while dataset B requires only $7ms$. Depending on the type of workloads that the datasets produce, such requirements translate into cache hit ratio requirements since the disk drives of the system are usually not fast enough to support the requirements without help from the cache. In general, cache is a shared resource among all datasets. Not all datasets are equally capable of utilizing the cache. The challenge then becomes to provide appropriate portions of the cache to the different datasets, taking into account, both their needs and their ability to utilize the resources made available to them. We also need to do so in a way which is relatively easy to implement. One of the methods which have been proposed to deal with QoS is cache partitioning, (B. Laliberte 2009), in which various portions of the cache are dedicated to various subsets of data. The problem is that partitioning is not flexible and may come at the expense of lowering the hit ratio.

A third challenge is to decrease the overall miss ratio of the cache.

In the present paper we consider the challenge of providing QoS, while decreasing the overall miss rate of the cache.

For implementing a QoS policy, while also improving the overall miss ratio of a cache, we use a simple soft mechanism that allows more time in the cache to datasets that have a proven track record of being able to utilize the cache and have a high priority. Each dataset gets a score which is based on these two criteria, and is inserted into the LRU queue in a position which depends on the score. This framework is very flexible and can easily accommodate changes to the criteria

that one applies when deciding on the overall importance of a data set.

Another important industry trend is that enterprise storage systems have large DRAM caches. More recently, flash drives have been introduced into enterprise storage systems. Flash drives are cheaper than DRAM and perform a bit more slowly, however, they are much faster than disk drives especially when servicing non-sequential I/O requests.

This state of affairs has created the opportunity of having a two tiered cache, consisting of a smaller DRAM portion and a larger flash portion. The issue of managing such a two tiered cache is not new and several suggestions have been made, see (E. Bachmat 1999; T. Wong and J. Wilkes 2002; I. Koltsidas and S. Viglas 2008). We evaluate some basic strategies with our production trace data. The results show that aggressively prefetching data in the second tier leads to substantially improved hit ratios. This is in line with the results of previous studies.

Finally, we combine our framework with the two tiered strategies and show that the results of the combination are better than either alone, both in terms of QoS and overall hit ratios.

We evaluate our framework with trace data from 5 real production systems. Detailed results are provided for the system which was most active and displayed the most complex workload. This system also displayed the most significant improvement using our framework. All 5 systems displayed some improvement in terms of hit ratio and response time showing that the method is stable and robust.

The paper is organized as follows:

Section 2 is devoted to an explanation of the components and data organization of an enterprise storage system.

In section 3 we introduce and evaluate our framework for improving hit ratios and providing QoS. We evaluate the hit ratio improvement portion against a standard LRU implementation. We then evaluate the addition of QoS considerations.

In section 4 we evaluate some basic algorithms for two layered caches, with our real production traces.

In section 5 we combine the methods of sections 3 and 4 and show improvement over both.

The work is part of a larger project that utilizes a similar framework to optimize all components of the storage system, not just the cache. As part of the larger project, the authors also worked on data configuration of secondary storage devices. Results for that portion of the project appeared in (G. Lipetz et al. 2013).

1.1 Related Work

There is an extensive literature on cache management schemes. Some basic general references include (E. Coffman and P. Denning 1973; O.I. Aven et al. 1987), where methods for improving the hit ratio such as LRU, LFU, cascading LRU, working set and many others have been considered. Cache partitioning for QoS has been considered, however, as noted above, this method is not as flexible and lowers the overall hit ratio. To the best of our knowledge a framework which simultaneously addresses both goals has not been suggested.

Issues which are related to two tiered caching have been considered in (E. Bachmat 1999; T. Wong and J. Wilkes 2002; I. Koltsidas and S. Viglas 2008) and section 4 is the first to consider these suggestions on trace data from real enterprise level production systems.

The use of flash drives in enterprise storage systems has been considered in (J. Gray and B. Fitzgerald 2008; S.R. Hetzler 2008; I. Koltsidas and S. Viglas 2008; S. Lee and B. Moon 2007; S.W. Lee et al. 2008; A. Leventhal 2008; E. Miller et al. 2001; M. Moshayedi and P. Wilkison 2008; D. Narayanan et al. 2009; S. Nath and A. Kansal 2007; G. Alvarez et al. 2001).

2. Background And Preliminaries

We provide a brief general description of the architecture of the type of storage systems which concern us in this paper. The main physical system components include directors, cache memory and secondary storage devices (disk drives). Flash drives are intermediate between secondary storage and cache. In this work we consider them as secondary cache components.

2.1 Components

The computational heart of the storage system is a set of CPUs called directors, which manage incoming host requests to read and write data and direct these requests to the appropriate storage components, either cache or secondary storage, which actually keep the data.

Cache memory (DRAM) is a fast and expensive storage area. The cache (DRAM) is managed as a shared resource by the directors. The content of the cache is typically managed by a replacement algorithm which is similar to FIFO or the Least Recently Used (LRU) algorithm. Data can be prefetched in advance of user requests if there is a good probability that it will be requested in the near future. Additionally, some data may be placed permanently in cache if it is very important, regardless of how often it is used. Whatever data is not stored in cache, resides solely on secondary storage. Typically, the DRAM cache comprises a very small portion of the total storage capacity of the system, in the range of $0.1 - 1$ percent.

Four basic types of operations occur in a Storage system: Read Hits, Read Misses, Write Hits, and Write Misses. A *Read Hit* occurs on a read operation when all data which is needed to satisfy the host I/O request is in cache. The requested data is transferred from cache to the host.

A *Read Miss* occurs when not all the data which is needed to satisfy the host I/O request is in cache. A director stages the block(s) containing the missing data from secondary storage and places the block(s) in a cache page. The read request is then satisfied from the cache.

The cache is also used for handling write requests. When a new write request arrives at a director, the director writes the data into one or more pages in cache. The storage system provides reliable battery backup for the cache (and usually also employs cache mirroring to write the change into two different cache boards in case a DRAM fails), so write acknowledgments can be safely sent to hosts before the page has been written (destaged) to secondary storage. This allows writes to be written to secondary storage during periods of relative read inactivity, making writing an asynchronous event, typically of low interference. This sequence of operations is also called a *write hit*.

In some cases the cache fills up with data which has not been written to secondary storage yet. The number of pages in cache occupied by such data is known as the *write pending count*. If the write pending count passes a certain threshold, the data will be written directly to secondary storage, so that cache does not fill up further with pending writes. In that case we say that the write operation was a *write miss*. Write misses do not occur frequently, as the cache is fairly large on most systems. A write miss leads to a considerable delay in the acknowledgment (completion) of the write request.

Not every write hit corresponds to a write I/O to secondary storage. There can be multiple write operations to the same page before it is destaged to secondary storage, resulting in write *folding*. Multiple logical updates to the same page are folded into a single destaging I/O to secondary storage. We will call a write operation to a secondary storage device a *destage write*.

The *read hit ratio* is the percentage of reads that were hits among all reads and similarly, the *write hit ratio* is the percentage of all writes which are hits. The *hit ratio* is the total number of I/O which were hits among all I/O.

2.2 Storage devices

There are several types of storage devices, typically used in enterprise storage systems. There is DRAM memory, Flash drives, FC derives and SATA drives. FC drives are typically 10K or 15K RPM spindles and SATA devices are typically 5400 or 7200 RPM, their seek times are measured in miliseconds, typically about 10ms for a SATA drive and 5ms for an FC drive. Flash drives are substantially faster than either FC or SATA drives, especially on non-sequential I/O. We will consider flash drives as an extension of the cache, since they are typically more similar in performance and throughput to DRAM rather than disk drives. Flash drives also have no moving parts and thus their performance is much less sensitive to changes in workload behavior and to the locations of I/O requests. In terms of read/write rate is the speed at which bytes can be read from the disk. For rotating disks (FC and SATA) the read/write rate of sequential data, is relatively stable and relatively close to the speed of flash drives (as compared to the difference in non sequential access overhead between SSDs and rotating disks). Flash drives have the highest throughput, although it is lower for writes than reads. This is because SSDs do not support random rewrite operations, instead they must perform *erasure* on a large segment of memory (a slow process), and can only *program* (or set) an erased block.

In terms of performance, we can think of the storage system as being tiered, with DRAM cache as the top performer (also most expensive), then the SSD drives forming a second cache tier, the FC drives are the top tier among disk drives, followed by SATA.

2.3 Logical Units and Extents

The data in the system is divided into user defined units, which are called logical units (LU) or volumes. An LU will typically span several GB. The LU is a unit of data which is referenced in the I/O communication between the host and the storage system. From the point of view of the storage system, the LU is divided into smaller units called *Extents*, which can be viewed as atomic units for storage management.

2.4 Verification of The Cache Management Policies

We need to verify and measure the performance of various management policies and algorithms. Since we do not have an enterprise level storage array, we developed a detailed simulation of a generic enterprise level array. Experience with a similar type of simulator which was employed during the design of Symoptimizer, (R. Arnan et al. 2007), a commercially available optimization product, shows that such simulators provide reliable indications of the improvement that can be expected in an actual system. We then coupled the simulation with several real traces of data from a production array to create a realistic test environment. The simulation was designed with the storage components and logical units described above. It includes a simulation of the cache and the other storage components, although we will mostly use the portion of the simulation which is related to the cache.

2.5 Trace data

The simulation uses real production trace data collected from several EMC Symmetrix systems from different customers. This data is hard to collect and requires special permission from the customer. Thus, the amount of trace data is limited.

The traces that we use in the simulations are composed of items which describe each and every I/O request during an extended time period. The items corresponding to each I/O request consist of:

1. time stamp: indicating when the request was received.
2. I/O operation: read or write.
3. logical unit ID: the volume targeted by the I/O request.
4. block offset: the offset of the start of the request within the logical unit

5. size: size of the I/O request, in blocks.

For example, a request might be to read 32 blocks from logical unit number 41, starting with block 15360 within the logical unit.

2.6 Cache

The cache is composed of cache blocks (pages) of a fixed size that typically is in the from $8 - 64$ KB range. For the purposes of the present work we will concentrate on the read portion of the cache and will assume that it is of a fixed size.

3. Cache Management Policies

The simplest policy to manage the cache is Least Recently Used (LRU). We will consider LRU as the default cache management policy and use it as a benchmark for the evaluation of our suggested management policies which we now describe.

3.1 The Hit Ratio LRU (HR LRU) Policy

When the cache contains data that the user does not access frequently, then the cache is under-utilized. We want to decrease the time that such data will reside in cache. One way to do so, is to gather statistics over each data set and observe how each one of them utilizes the cache. Since gathering statistics over each block is too specific and may lead to over-fitting, we gather statistics for data units which we will call *volume chunks*. The volume chunk, or chunk for short, is a portion of data which can be as large as a whole LU, or a smaller data set of size 20MB to 100MB. Note that this is much larger than the cache page size that will vary from 8KB to 64KB in our evaluations. The chunk will serve as an atomic unit for the management policy. If the cache hit ratio of the *volume chunk*'s I/O is low then we would like to avoid leaving it inside the cache for a long time.

We first consider a cache management policy that takes into account the chunk's hit ratio and decides accordingly where to place the data in the LRU queue. Chunks which are rarely used will be inserted towards the middle or near the head of the queue and thus will be removed from the cache via the LRU mechanism in less time. This policy does not take QoS considerations into account. For fairness data will never be inserted too close to the head of the queue.

We model the cache as an LRU managed queue of cache pages where the content of the pages comes from the different chunks. A position in the queue is given by a percentage. For example, a 30% position means that 30% of the cache pages are queued between that position and the head of the queue. A large percent corresponds to a position near the tail, hence the data in such a position will reside in cache for a relatively long time period.

The caching algorithm work as follows:

When we want to insert new data which belongs to a certain chunk and the cache is full, we remove the page (or pages) from the head of the queue and insert the data into a position in the queue according to the following prescription.

Let HR be the hit ratio of the chunk containing the data (in percentage points).

Let $MinPosition$ be the smallest position in the cache that data may be placed in.

Let $MinHR$ be an adjustment factor.

$HRPosition \leftarrow$ min(HR + $MinHR$, 100%) of cache size.

Final position $\leftarrow$ max($MinPosition$, $HRPosition$).

According to this formula we can see that each data receives a position of at least $MinPosition$ of the cache size and can be placed deeper in the cache queue based on the data chunk's hit ratio. The higher the hit ratio, the deeper in the queue the data is placed.

3.2 Approximate cache position

To implement the algorithm, we need to approximate cache positions. This can be achieved by dividing the cache LRU queue into k sub-queues (parts), numbered $1, \ldots, k$ via pointers to the heads and tails of the parts. The tail of part i is the successor of the head of part $i - 1$. Let p_i be the size of part i in terms of the number of pages. We will insist that for all i we have $|p_i - p_{i-1}| \leq 1$. By a telescopic sum, this implies $|p_i - p_j| < k$ for all $1 \leq i, j \leq p$.

We need to keep this balance when performing the basic operations of an LRU queue. These are

1) Inserting a new item into cache. In this case, the last part k loses one item and the first part gains one item.

2) After a read hit, moving the page that was hit to the head of the queue. This means that part i which contained the page which was hit loses a page, while part 1 gains a page.

3) Removal of a page from the queue, for example, when a page incurs a write it is moved to a different data structure which contains the dirty pages. In this operations, part i loses an item.

4) Insertion of an item into one of the parts of the queue in its head position. This means that part i has gained one item.

It is easy to verify that in all 4 cases the imbalances which may arise from these operations can be fixed in at most k steps. As an example, in case 3, one can return to the configuration before the change by shifting each of the pointers of parts $1, \ldots, i$, to the pointers predecessor, thus lowering parts, $1, \ldots, i-1$ by 1 and adding one item to part i. This also solves case 1 which is a special case of 3. The same holds in case of a gain or loss of a page by one of the parts. If a page was lost from part i then we find a $i < j$ such that $p_i = p_{i+1} - 1 = p_{i+2} - 2 = \ldots = p_j - (i - j)$ and either $p_j \geq p_{j+1}$ or $j = k$. We then transfer as before an item from j to i via $j - i$ pointer movements. A similar procedure is applied for a gain of a page.

If we consider an unbounded sequence of read miss operations, case 1, then we can easily see that any balancing scheme will have an amortized cost of at least $k - 1 - \varepsilon$ pointer changes, so the scheme is optimal in the worst case.

3.3 Priority LRU Policy

We would like to add QoS considerations into our framework. We assume that each LU, is given a priority which we denote by $prio(k)$, where k is the LU number. In general the priorities can have any positive values. The idea then is to assign to each piece of data from chunk j and LU $k(j)$ a cache position that will reflect the hit ratio of chunk j and the priority of LU k to which the data belongs. We now describe the specific scenario that we considered and the specific choice of position function that we used for the purposes of our evaluation. We considered a scenario with three priority levels where $prio(k) = 1$ for LUs in the high priority group, $prio(k) = 2$ in the medium priority group and $prio(k) = 3$ in the low priority group. All the LUs were assigned to one of these three priorities. Note that an LU is larger than a chunk which is larger than a cache page. We will see that our framework can handle these diverse scale levels.

We would like to improve the response time of the high priority data by prolonging its stay inside the cache while avoiding holding in the cache data that is rarely being accessed. The position of newly inserted data inside the cache is decided by the following algorithm:

Let $prio(data) = prio(k(j))$ be the data's priority group.

Let $HR(data) = HR(j)$ be the hit ratio of the chunk (j) containing the data (in percentage points).

Let $priorityWeight$ be a constant that represents the impact the priority has on the position of the data.

prioCoeff(data)$\leftarrow 1 - \frac{prio-1}{5}$

PrioPosition(data)$\leftarrow$min(($priorityWeight$*(3-$prio(data)$))% + $HR(data) * prioCoeff(data)$, 100%) of cache size

FinalPosition$\leftarrow max(MinPosition, PrioPosition(data))$

The $PrioPosition$ calculation is composed of two parts. The first is defined solely by the data's group priority, each data from the highest priority group receives $2 * priorityWeight\%$ of cache size, data from the medium priority group receives $priorityWeight\%$ of cache size while data from the lowest priority group does not receive any contribution from this part. The second part takes into account the hit ratio of the data's volume chunk and the data's group priority, for data from the high priority group this part adds its hit ratio as it is, for data from the medium priority group this part adds 80% of the data's hit ratio and for data from the low priority group this part adds only 60% of its hit ratio to the position calculation. As in the Hit Ratio Policy case each data receives at least $MinPosition$ of cache size. We note that the algorithm easily adjusts to having more than 3 priority groups and to any choice of monotone relation between the overall weight of a certain dataset and the position into which it is inserted. The above pseudocode, merely represents the choices that we have evaluated. We also note that the results are fairly robust in the sense that any reasonable choice of monotone function which sufficiently differentiates between the priority groups seems to give similar results.

3.4 Algorithm Flow

The simulation runs in time units. For the purposes of the evaluation the time unit was chosen to be 1 hour. During each time unit the simulation receives all the trace information and simulates the I/O requests in the system. For every I/O we simulate the director who checks whether the requested section of the extent exists in the cache. If it does, the I/O is considered a hit. If it does not exist, it is considered a miss and is brought into the cache based on the cache management policy from the storage component it resides in. Throughout the simulation, we aggregate statistics such as hit ratio of each *volume chunk* and miss ratio for each priority group every hour and update these fields.

While handling each I/O we also calculate its response time. The response time takes into account the amount of time it takes the director to search the cache for the required section. When the I/O is a miss, the time it takes to fetch the requested section from the relevant secondary storage device (disk drive) is added to the response time.

3.5 Experimental Evaluation

In this section we describe the experiments conducted to evaluate the HR LRU and the Priority LRU cache management policies. Our experiments are designed to examine:

- The improvement in each priority group's miss ratio.
- The overall improvement of the miss ratio in the system.

3.6 Comparison

We use the LRU policy as a benchmark in order to examine both the HR LRU and the Priority LRU cache management policies, by comparing the miss ratio for each priority group and the miss ratio for the system in total. We ran several simulations on each data trace. The simulations ran several times, each time with a different policy. The first group of simulations ran with *volume chunks* coinciding with LU's, i.e. hit rate data was aggregated at the LU level. The second group of simulations ran based on 20MB extents as *volume chunk*, with a hit ratio now assigned to a specific 20MB extent that the data resided in. Each group was tested with cache page sizes of 8 KB and 64 KB.

3.7 Analysis

We had data from 5 customers. The data we present in detail is from one customer (C5) over a 24 hour period. This customer displayed the highest level of activity and the

most complex workload. Results from other customers will be described later on. This customer trace contains 187 LUs, each of size that varies from a few GB to hundreds of GB. The data is not being accessed uniformly along the course of the simulation as there are highly active periods, and more relaxed times, as shown in figure 1. We have divided the LUs into three priority groups the following way:

The High priority group represents 10% of the total number of LUs. The medium priority represents 30% and the low priority 60%.

We have fixed the following algorithm parameters:

$MinPosition = 20\%$ which means that each dataset will have a position of at least 20% of the cache for the HR LRU and the Priority LRU cache management policies.

$MinHR = 10\%$ we use this factor in order to give cache pages with relatively high hit ratios better positions in cache.

$priorityWeight = 15\%$ this parameter which is used in the Priority LRU policy.

In order to compare between the cache management policies, we ran simulations with the following system settings:

Cache Size: 5GB and 20 GB.

Cache page size: 8KB and 64 KB.

The first issue we want to examine is which *volume chunk* size will give us better performance. When we ran simulations with 100MB as the *volume chunk* size the results were similar to those using the whole LU as *volume chunk* size. consequently, we ran only two sets of simulations for each trace, one with an LU being the *volume chunk* size and the second with 20MB as the *volume chunk* size.

We can see in figure 2 that the simulation with chunks of size of 20 MB gave us a modest improvement in the miss ratio of the cache throughout each time unit in the simulation in comparison to the simulation with an LU as *volume chunk* size. The overall improvement for the total miss ratio of the cache was around 4%.

In addition, we ran the simulations comparing 20MB as *volume chunk* size and LU as *volume chunk* size on all other customers and the results have shown that having a 20MB chunk size as a volume chunk improves the miss ratio of the cache by 1% to 4%. The improvement is expected since the data which is used for hit ratio calculations is more accurate and detailed in this case. In fact its a little surprising that the granularity level of an LU is for the most part sufficient. Nonetheless, since 20MB chunks perform better, the results led us to using the 20MB chunks for all further experiments.

We also ran several simulations with primary cache composed of cache blocks of size 8KB and 64KB. As can be seen from figure 3 there was only a slight improvement in all the cache management policies for the systems with cache block size of 64KB. The reason for the improvement is that the average I/O size tends to be bigger than 8KB and a page of size 64KB leads to more total hits by bringing in a bit more data. Following our results, we have decided to run our simulations only with 64KB cache page size.

Figure 4 describes the cache miss ratio of the two sets of simulations as mentioned earlier. In graphs (a), (b) and (c) we can see the cache miss ratio for each group and for each cache management policy with 5GB primary cache. We can also observe that the miss ratio for the high priority group decreases from around 55% for the LRU and HR LRU policies to around 40% for the Priority LRU policy. In the medium priority group there is a slight improvement with the Priority LRU policy and for the low priority group this policy caused only slight damage.

As can be seen from table 1, the overall improvement achieved by the Priority LRU policy over the LRU policy for the high priority group is above 47% for both 5GB and 20GB cache sizes. Thus, we achieve the main goal of the Priority LRU algorithm, to obtain nearly optimal QoS performance. In addition, we can see that although the policy gives the priority of the data a major weight in its calculation, it still achieved between 4% to 8% improvement in the cache miss ratio of the system over LRU.

We notice that for the 5GB cache simulation we achieved only a slight improvement for the medium priority group, while in the 20GB cache simulation we have achieved an improvement of almost 15% in the total miss ratio of the cache. This underlines the conclusion we described before that the Priority LRU policy performance is dependent on the size of the cache, in the larger cache, the medium priority group had enough cache to also benefit from the additional

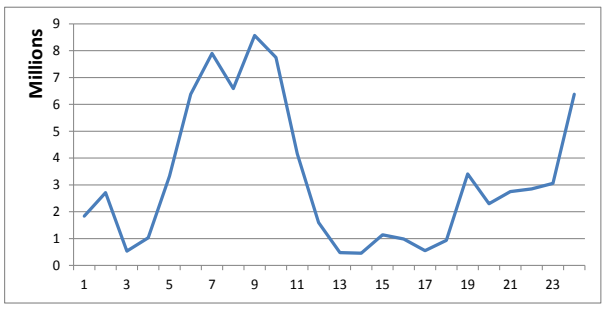


Figure 1. I/O distribution throughout the simulation.

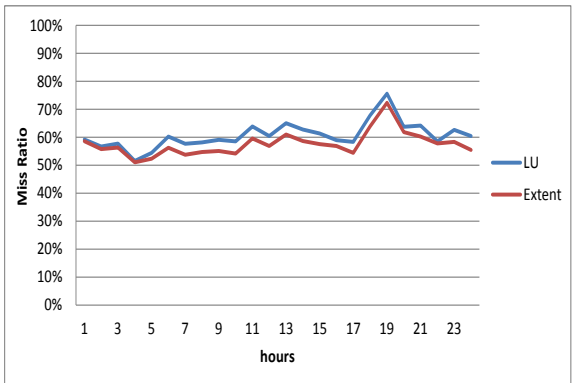


Figure 2. Miss ratio of the cache for simulations with 20MB as volume chunk size and LU as a volume chunk size.

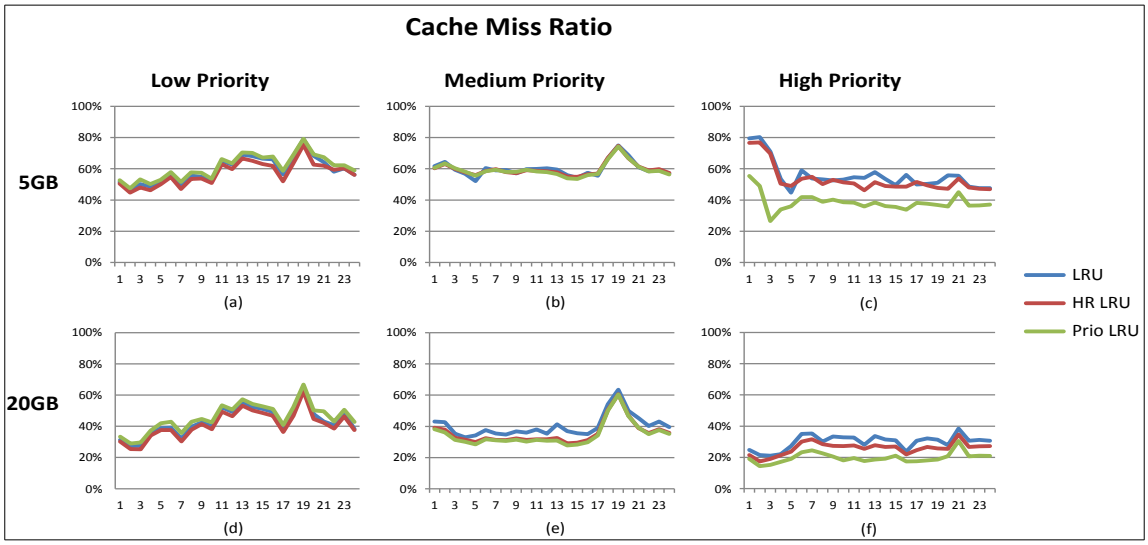


Figure 4. Miss ratio for each priority group and cache size.

cache space it received at the expense of the low priority group.

In table 2 we can see the summary of the improvements gained by Priority LRU and HR LRU policies over the LRU policy for multiple customers. As we can see, we have achieved a significant improvement in total miss ratio for the simulations with the HR LRU policy over the LRU policy.

We have also successfully achieved an improvement in the QoS, i.e. the high priority group's miss ratio was significantly decreased with the Priority LRU cache management policy. We can also observe that in most cases we have achieved not only a QoS improvement, but also an overall system miss ratio decline of several percents.

4. Two Layered Cache Model

4.1 Two Layered Cache Management Policies

In this section we evaluate two-level caching policies, such as the ones considered in (E. Bachmat 1999; T. Wong and J. Wilkes 2002; I. Koltsidas and S. Viglas 2008), using our trace data. These methods will be combined with those of the previous section in the next section of the paper.

Our basic set-up is a primary DRAM cache, followed with a secondary flash cache which may be managed differently. The page sizes also need not be the same, with the DRAM page size. We will consider the case where the DRAM page size is smaller than the flash page size.

For every I/O, we simulate the director who checks whether the requested section of the extent exists in the cache. If it does, the I/O is considered a *cache hit*. If it does not exist, the director checks whether it resides in the flash cache, if it does, the I/O considered as a *flash cache hit*. If it does not exist in the flash cache, it is considered a miss and is brought into the cache from the storage component it resides in according to the cache management policy, as follows:

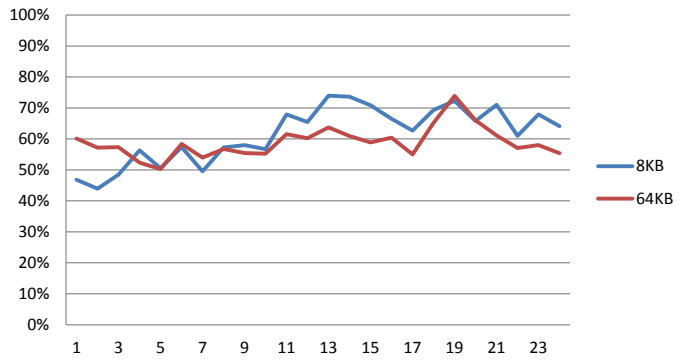
- Layered policy - When the data of an I/O does not exists in both primary cache and flash cache, we fetch the data from the device that it resides in to the primary cache. When the data needs to be inserted to the primary cache and it is full, we empty a DRAM page according to the cache removal policy (LRU). The removed page from the first tier is inserted into the secondary flash tier. If the page size in the flash tier is larger, which is the case in our implementation (1MB), we bring the remaining aligned data from the secondary storage to fill the flash page. If the second tier cache is full before the described insertion, we remove one of the other flash pages according to the removal policy of the second tier cache.
- Dual policy - When the data of an I/O does not exists in both primary cache and flash cache, we bring the 1MB aligned data from the device that it resides in and insert the relevant data piece (8KB or 64KB) into the DRAM cache and the whole 1MB data to the flash cache. When either the primary cache or the flash cache is full, before we insert the data, we remove older data from the relevant cache according to the cache removal policy.

	5 GB Cache				20 GB Cache			
	High	Medium	Low	Total	High	Medium	Low	Total
LRU	57.28%	60.76%	58.19%	59.16%	28.90%	41.47%	42.05%	39.77%
HR LRU	54.46%	60.51%	56.12%	57.77%	25.26%	37.07%	40.08%	36.48%
Priority LRU	38.90%	60.23%	59.99%	56.84%	19.54%	36.23%	44.42%	37.02%
Priority LRU Improvement	47.25%	0.88%	-3.01%	4.08%	47.91%	14.46%	-5.32%	7.43%

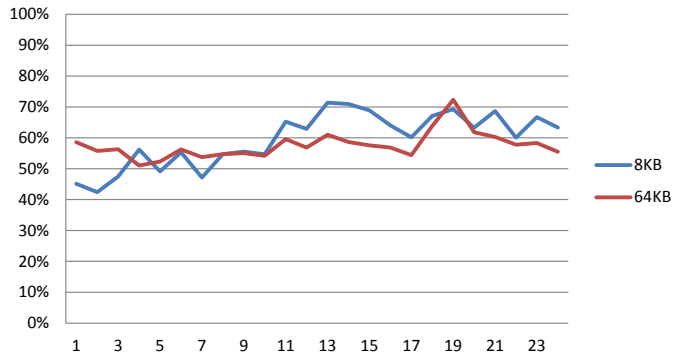
Table 1. Total miss ratio table for each priority group and cache size

	5 GB Cache					20 GB Cache				
	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
C1	0.5152	0.5083	0.5074	1.36%	1.54%	0.3506	0.3563	0.3477	-1.60%	0.83%
C2	0.7949	0.7730	0.7673	2.83%	3.60%	0.6656	0.6398	0.6380	4.03%	4.33%
C3	0.4347	0.4311	0.4301	0.84%	1.07%	0.3964	0.3908	0.3916	1.43%	1.23%
C4	0.3444	0.3338	0.3331	3.18%	3.39%	0.2917	0.2860	0.2854	1.99%	2.21%
C5	1.0328	1.0091	0.9504	2.35%	8.67%	0.6673	0.6040	0.5881	10.48%	13.47%

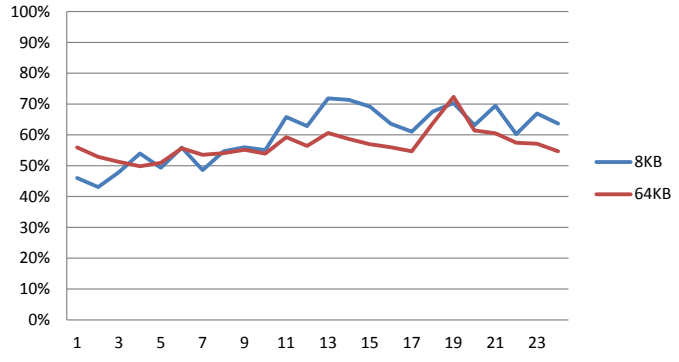
Table 3. All priority groups weighted miss ratio summary table over all customers



(a) LRU Miss Ratio



(b) HR LRU Miss Ratio



(c) Priority LRU Group Miss Ratio

Figure 3. Miss ratio comparison between 8KB and 64KB cache pages

(a) High Priority Group Miss Ratio

	5 GB Cache					20 GB Cache				
	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
C1	46.77%	45.89%	45.31%	1.92%	3.21%	35.06%	35.23%	32.51%	-0.47%	7.85%
C2	27.36%	24.96%	22.42%	9.63%	22.03%	19.64%	19.01%	18.20%	3.34%	7.94%
C3	16.93%	16.86%	16.71%	0.44%	1.35%	16.01%	15.93%	15.86%	0.48%	0.91%
C4	12.51%	11.25%	10.50%	11.24%	19.20%	8.78%	8.22%	7.86%	6.84%	11.79%
C5	57.28%	54.46%	38.90%	5.17%	47.25%	28.90%	25.26%	19.54%	14.42%	47.91%

(b) Medium Priority Group Miss Ratio

	5 GB Cache					20 GB Cache				
	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
C1	20.98%	20.40%	20.38%	2.86%	2.98%	13.10%	13.20%	13.18%	-0.75%	-0.62%
C2	41.79%	40.69%	40.60%	2.71%	2.93%	34.24%	32.54%	32.47%	5.22%	5.44%
C3	22.53%	22.19%	21.99%	1.49%	2.43%	19.87%	19.45%	19.52%	2.15%	1.78%
C4	21.88%	21.59%	21.37%	1.34%	2.37%	19.50%	19.24%	19.13%	1.38%	1.96%
C5	60.76%	60.51%	60.23%	0.41%	0.88%	41.47%	37.06%	36.23%	11.89%	14.46%

(c) Low Priority Group Miss Ratio

	5 GB Cache					20 GB Cache				
	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
C1	44.00%	44.31%	44.54%	-0.69%	-1.21%	30.73%	30.02%	31.42%	2.37%	-2.19%
C2	65.89%	65.63%	66.50%	0.41%	-0.91%	63.52%	63.23%	63.75%	0.45%	-0.37%
C3	37.72%	37.56%	37.98%	0.42%	-0.67%	34.67%	34.30%	34.47%	1.07%	0.58%
C4	30.00%	29.60%	30.36%	1.35%	-1.18%	26.37%	26.14%	26.46%	0.88%	-0.35%
C5	58.19%	56.12%	59.99%	3.68%	-3.01%	42.05%	40.08%	44.42%	4.93%	-5.32%

(d) Total Miss Ratio (over all priority groups)

	5 GB Cache					20 GB Cache				
	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement	LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
C1	32.68%	32.46%	32.57%	0.68%	0.34%	22.25%	22.35%	22.31%	-0.45%	-0.29%
C2	45.97%	45.16%	45.12%	1.79%	1.88%	39.56%	38.93%	38.83%	1.62%	1.88%
C3	27.20%	26.99%	27.04%	0.59%	0.78%	24.77%	24.45%	24.52%	1.31%	1.02%
C4	20.47%	20.25%	20.42%	1.09%	0.24%	21.60%	21.41%	21.52%	0.89%	0.37%
C5	59.17%	57.77%	56.84%	2.42%	4.10%	39.77%	36.48%	37.02%	9.02%	7.43%

Table 2. Miss ratio summary tables over all customers

Basically the dual policy is a form of prefetching, where the prefetched data is placed in the secondary flash cache. Such prefetching makes sense since the data that is prefetched into the secondary cache is sequential to the data that was required and therefore the prefetching process is efficient in secondary storage. For example, reading 1MB of data on a

4TB SATA drive takes about 3ms. In contrast, the seek and rotational latency time is roughly 10-12 ms. Thus, bringing 1MB of data instead of 64K increases the utilization of the secondary storage device by roughly 20-25% . If the device is not highly utilized then this will lead to a small incremental addition to the average service time of the secondary device, which may easily be offset by the hit ratio gains.

5. Two Tiered Cache with Cache Management Policies

In this section we will combine the Two Tiered cache model along with its cache management algorithms with the QoS cache management policies.

5.1 Experimental Evaluation

In this section we describe the experiments conducted to evaluate the two tiered cache model with the cache management policies. Our experiments are designed to examine:

- The improvement in the total and per group miss ratios gained by the Dual policy over the Layered policy when combining it with the HR LRU and the Priority LRU cache management algorithms.
- The improvement in the system's miss ratio when using the HR LRU and the Priority LRU cache management policies in the Two Layered cache model environment.
- The improvement in the average response time of the system when using the HR LRU and the Priority LRU cache management policies in the Two Layered cache model environment.

5.2 Comparison

In order to compare between the cache management policies in the Two Layered cache model environment, we ran simulations with the following parameter values:

Primary cache page size of 64KB.
Primary cache size of 5GB.
Flash cache page size of 1MB.
Flash cache size of 50GB.

We chose those values because they gave us the best results for the previous simulations, which we have run when evaluating the Two Layered cache model and the QoS cache management policies separately, as described in the earlier sections.

During each time unit we have calculated statistics about the miss ratio in both the primary cache and the flash cache.

5.3 Analysis

For each customer trace we ran simulations with *Layered* and *Dual* policies in combination with the *LRU*, *HR LRU* and *Priority LRU* cache management algorithms. As before, the data we are about to present in detail is from customer C5 that we discussed in section 3.

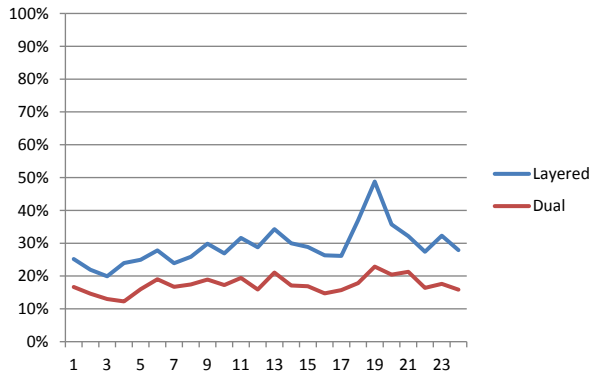


Figure 5. HR LRU policy miss ratio Layered vs Dual

Miss Ratio Summary Table

		LRU	HR LRU	Priority LRU	HR LRU Improvement	Priority LRU Improvement
High Priority Group	Layered	20.06%	17.06%	14.13%	17.57%	41.97%
	Dual	12.69%	10.60%	9.18%	19.69%	38.25%
	Dual Improvement	58.12%	60.97%	53.98%		
Medium Priority Group	Layered	33.64%	30.39%	28.90%	10.69%	16.39%
	Dual	21.65%	18.83%	17.25%	15.01%	25.54%
	Dual Improvement	55.37%	61.43%	67.58%		
Low Priority Group	Layered	33.22%	32.05%	34.96%	3.65%	-4.96%
	Dual	19.06%	17.98%	20.34%	6.01%	-6.25%
	Dual Improvement	74.26%	78.24%	71.90%		
Total System	Layered	31.37%	29.01%	29.11%	8.12%	7.78%
	Dual	19.20%	17.21%	17.27%	11.59%	11.21%
	Dual Improvement	63.36%	68.60%	68.55%		
Total System Weighted	Layered	52.19%	47.50%	46.04%	9.87%	13.36%
	Dual	32.54%	28.67%	27.60%	13.50%	17.90%
	Dual Improvement	60.39%	65.68%	66.81%		

Table 4. Miss ratio summary table

As can be seen in figure 5 and table 5, the HR LRU cache management policy improves its performance significantly when combined with the Dual policy. The improvement over the basic LRU policy combined with either the Dual or layered policy is about 10%, i.e., the improvement is independent of the improvement gained by adding the secondary flash cache and is largely independent of the management of the additional secondary cache (Dual or Layered).

The improvements gained by the HR LRU and Priority LRU policies when combined with the Layered and the Dual policies were somewhat similar, thus we chose to discuss only the improvements gained when using the Dual policy.

In table 4 we can observe a significant improvement for all the cache management algorithms, over all the priority groups, of the Dual policy over the Layered policy. These improvements range from around 55% to around 80%. We can also see that we have obtained an improvement of 8% for

Miss Ratio Summary Table

		HR LRU Improvement	Priority LRU Improvement				
			High Priority Group	Medium Priority Group	Low Priority Group	Total System	Total System Weighted
Customer 1	Layered	1.61%	0.43%	0.69%	-1.84%	1.86%	2.52%
	Dual	1.96%	1.87%	1.57%	-2.89%	1.25%	2.24%
Customer 2	Layered	2.15%	8.46%	3.10%	-0.81%	1.87%	5.66%
	Dual	2.51%	17.40%	3.75%	-1.21%	2.43%	7.67%
Customer 3	Layered	2.92%	6.99%	2.55%	0.63%	1.39%	3.03%
	Dual	2.48%	10.10%	2.24%	-1.52%	2.95%	8.14%
Customer 4	Layered	1.17%	7.98%	2.90%	-1.02%	0.32%	1.07%
	Dual	0.26%	8.86%	3.46%	-1.92%	-0.39%	1.83%
Customer 5	Layered	8.12%	41.97%	16.39%	-4.96%	7.78%	13.36%
	Dual	11.59%	38.25%	25.54%	-6.25%	11.21%	17.90%

Table 5. All customers miss ratio summary table

the HR LRU policy with the Layered policy and around 12% with the Dual policy. For the Priority LRU policy we have achieved a significant reduction in miss ratio of around 40% with both Dual and Layered policies for the high priority group, between 16% to 25% for the medium priority group and a worsening of around 5% for the low priority group, but still there was an overall improvement of around 8% for the Layered policy to around 11% for the Dual policy. We notice that although the Priority LRU policy takes into account QoS, which can lead to a bad overall performance, for this particular customer we still achieve an improvement close to the one gained by the HR LRU policy.

In table 5 we can see a summary of all the results for all the customers. This table is composed of two parts, the first is the improvement gained by the HR LRU over the LRU policy in the total miss ratio and the second is the improvement achieved by the Priority LRU policy over the LRU policy for each priority group and for the total system.

We can see two contradictory trends that occur for different customers in the comparison between the Priority LRU policy and the LRU policy.

The first happens for the third customer. We have achieved a positive improvement for the low priority group. This can be explained by the features of the customer's data and the Priority LRU policy's cache page position formula. This formula consists of two main components, the first is the hit ratio of the extent the data resides in and the second is the priority group the data belongs to. If the average hit ratio of the low priority group extents is relatively high, then the priority component has less impact on the position of the block in the cache, which leads to a low miss ratio, and because the LRU policy does not take into account the hit ratio, we achieve an improvement in the low priority group's miss ratio. We can also notice that for the Dual policy there was a worsening in the miss ratio of the low priority group. This happens due to the significant improvement the Dual policy achieves, thus leading to a hit ratio weight reduction in the Priority LRU policy's position formula, while the priority weight grows, and that leads to a higher miss ratio, hence we have a worsening in the low priority group stats for the Priority LRU policy.

The opposite phenomena occurs for the fourth customer. We can see that for the Dual policy the total miss ratio achieved by the Priority LRU was worse than the one achieved by the LRU policy. This can be explained by the fact that the low priority group has more weight in the total miss ratio calculation in comparison to the other priority groups. Moreover, the HR LRU policy achieves only a slight improvement over the LRU policy, which leads to a conclusion that this data is relatively sparse among the LUs, thus the hit ratio component in the position formula is less significant and this leads to high miss ratio for the low priority group.

Similar results were obtained when comparing the response time stats for all the policies over all the customers.

6. Conclusions and future work

In this paper we introduced cache management policies which take into account both the use of cache resources by the data and the QoS requirements of the data. The first policy, the HR LRU policy ignores QoS requirements and improves hit ratios by giving more cache space to data that uses the cache more efficiently. The second policy, the Priority LRU policy does the same but also takes into account the importance of the data. These policies also improve hit ratios, when combined with two tier management policies for handling flash based cache extensions. The amount of improvement depends on the workload of the customer, but the algorithms are robust in the sense that they improve performance across all customers that were tested.

As part of our investigation we also compared the performance of two methods (Dual and Layered) for managing two tier caches on real customer workloads. We have seen (in agreement with previous studies) that the Dual approach which amounts to aggressively prefetching data into the secondary tier is superior.

In future work we plan to consider more refined criteria for data to obtain more cache, i.e., instead of the hit ratio we will consider estimates of the derivative of the hit ratio given the current cache size. We also plan to bring together the caching algorithms of the present paper with the back-end placing algorithms of (G. Lipetz et al. 2013), to present an end to end optimization scheme for improving storage system performance in a QoS setting.

References

M. Allalouf, Y. Arbitman, M. Factor, R.I. Kat, K. Meth, and D. Naor, Storage modeling for power estimation, in proceedings of The Israeli Experimental Systems Conference (SYSTOR'09), May 4-6, 2009, Haifa, Israel

G. Alvarez, E.Borowsky, S.Go, T. H. Romer, R. Becker- Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. ACM Transactions on Computer Systems, Vol. 19, 483 - 518, 2001.

D. Anderson, J. Dykes, and E. Riedel. More than an interface - SCSI vs. ATA. In Proc. USENIX Conference on File and Storage Technologies (FAST), 245-257, San Francisco, CA, March 2003.

R. Arnan, E. Bachmat, T.K. Lam and R. Michel, Dynamic data reallocation in disk arrays, ACM transactions on storage, vol 3(1), 2007.

O.I. Aven, E.G. Coffman and Y.A. Kogan, *Stochastic analysis of computer storage*, D.Reidel publishing, 1987.

E. Bachmat, T.K. Lam, and A. Magen, Analysis of set-up time models - a metric perspective, Theoretical computer science, vol 401, 172-180, 2008.

E. Bachmat, Caching system and method providing aggressive prefetch, US Patent 6,003,114, 1999.

E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes. Capacity planning with phased workloads. In 1st Workshop on Software and Performance (WOSP'98), pages 199-207, Santa Fe, NM, Oct 1998.

F. Chen, D. Koufaty and X. Zhang Understanding intrinsic characteristics and system implications of flash memory based solid state drives, in Proceedings of SIGMETRICS 2009, 181-192, 2009.

E. Coffman and P. Denning, Operating Systems Theory, Prentice-Hall, 1973.

E. Gal and S. Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2): 138-163, 2005.

J. Gray and B. Fitzgerald, Flash Disk Opportunity for Server Applications, ACM Queue, Vol. 6, Issue 4, 18-23, 2008.

J. Guerra, H. Pucha, J. Glider, W. Belluomini and R. Rangaswami, Cost Effective Storage using Extent Based Dynamic Tiering, Proc. of FAST 2011.

S.R. Hetzler, The storage chasm: Implications for the future of HDD and solid state storage. http://www. idema.org/, December 2008.

L. Kleinrock, *Queueing Systems. Volumes 1-2*, Wiley-Interscience, 1975.

I. Koltsidas and S. Viglas. Flashing up the storage layer. In Proc. International Conference on Very Large Data Bases (VLDB), pages 514-525, Auckland, New Zealand, August 2008.

B. Laliberte. Automate and Optimize a Tiered Storage Environment FAST! ESG White Paper, 2009.

S. Lee and B. Moon. Design of flash-based DBMS: An in-page logging approach. In Proc. of SIGMOD'07, 2007.

S.W. Lee, B. Moon, C. Park, J. Kim, and S. Kim, A case for flash memory SSD in enterprise database applications, *In Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1075-1086, Vancouver, BC, June 2008.

A. Leventhal. Flash storage memory. In Communications of the ACM, volume 51, July 2008.

G. Lipetz, E. Hazan, A. Natanzon, E. Bachmat, Automated tiering in a QoS environment using sparse data, Proceedings of HPCC13, 2013.

E. Miller, S. Brandt, and D. Long. HeRMES: High-performance reliable MRAM-enabled storage. In Proc. IEEE Workshop on Hot Topics in Operating Systems (HotOS), pages 95-99, Elmau/Oberbayern, Germany, May 2001.

M. Moshayedi and P. Wilkison. Enterprise Flash Storage, ACM Queue, Vol. 6, 32-39, 2008.

D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating enterprise storage to SSDs: analysis of tradeoffs. In Proc. of EuroSys'09, 2009.

S. Nath and A. Kansal, FlashDB: Dynamic self tuning database for NAND flash. In Proc. Intnl. Conf. on Information Processing in Sensor Networks (IPSN), pages 410-419, Cambridge, MA, April 2007.

M. Peters. Compellent harnessing ssds potential. ESG Storage Systems Brief, 2009.

M. Peters. Netapp's solid state hierarchy. ESG White Paper, 2009.

M. Peters. 3par: Optimizing io service levels. ESG White Paper, 2010.

Taneja Group Technology Analysts. The State of the Core Engineering the Enterprise Storage Infrastructure with the IBM DS8000. White Paper, 2010.

T. Wong and J. Wilkes, My cache or yours, Proceedings of USENIX technical conference , 161-175, 2002.