Integrated caching and tiering according to use and QoS requirements

Mark Abashkin
Department of Computer science
Ben-Gurion University of the Negev
Beer-Sheva, Israel, 8410501
Email: abashkin@cs.bgu.ac.il

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

Eitan Bachmat
Department of Computer science
Ben-Gurion University of the Negev
Beer-Sheva, Israel, 8410501
Email: ebachmat@cs.bgu.ac.il

***Abstract*—In this paper we consider the management of a tiered storage system consisting of disk and flash drive storage and a DRAM cache, with the challenge of taking into account heterogeneous quality of service (QoS) requirements.**

We integrate and adjust methods which control the use of fast resources such as flash drives and cache, according to the user access patterns of different data extents and the QoS requirements of the extents, which we developed in previous work. Using traces from real production systems, we show that the benefits of the integrated system are substantially larger than that provided by each method alone. Our method is able to substantially improve the performance of data with high QoS demands, with little or no damage to data with low QoS demands. Thus we are able to exploit the resources of the storage system to the advantage of all data types. we show improvements in the range of 9%-71% for datasets with the highest QoS requirements, and 0%-70% response time improvement overall, compared to a QoS optimized system which took only disk drive resource allocation into consideration. In workloads where cache is useful we obtain large gains, showing that it is important to integrate back-end (drives) and front-end (cache) optimization.

I. Introduction

Providing QoS is an important challenge in enterprise storage systems, especially in heterogeneous environments which are typical in our increasingly virtualized and cloud based computing world. Quality of service is usually defined in terms of some performance metric, such as IOPS or average response time. Different portions of the user dataset need to achieve different performance targets. Depending on the type of user access pattern (workload) of the datasets, such requirements translate into cache and flash residency requirements and load balancing requirements to avoid queues. The challenge then becomes to provide appropriate portions of the cache and flash drives to the different datasets, taking into account, both their needs and their ability to utilize the resources made available to them. it is also important that our resource management methods will be relatively easy to implement across many types of storage systems.

In previous work [13] we introduced an algorithm which allocated resources back-end resources (disk and flash drives) to different datasets according to their QoS requirements and their ability to take advantage of resources such as flash drives. It was shown that we can provide better performance to datasets with stringent QoS requirements without hurting too much (or not at all) other datasets.

In this paper we show that by taking into account the QoS requirements and the ability to take advantage of cache of different datasets we can provide further substantial improvements beyond those presented in [13]. Cache management is different from the management of drives, being more dynamic and real time driven, hence our caching algorithms have to be easy to implement and are different from the back-end algorithms. We use a simple mechanism that allows us to push data into different locations of the LRU cycle. Data belonging to datasets which have shown their capability of exploiting cache (good hit ratios) and have stringent QoS requirements (need high hit ratios) are injected into the beginning of the LRU cycle and will reside in cache for a lengthy time period. On the other hand, datasets which do not utilize cache well, or do not require good performance are injected near the tail of the LRU cycle and spend little time in cache before being removed, unless they are requested. This framework is very flexible and can easily accommodate changes to the criteria that one applies when deciding on the overall importance of a dataset.

In addition, the mechanism is integrated with the back-end resource allocation mechanism in an algorithm which adjusts the weights (priorities) of the different datasets according to how well we are achieving their QoS goals.

We evaluate our method with trace data from real production systems. The results of the combined backe-end/cache algorithm show substantial improvement over a back-end algorithm combined with a generic LRU algorithm. In all the workloads we have improvement in the performance of the QoS stringent datasets, however, the level of improvement varies substantially, between 9% and 71% . The reason is that in some systems the workload tends to be rather random and has difficulties exploiting the improved cache algorithms. For the same reason, the overall response time has improvement ranging from negligible to 70% . Workloads that can utilize cache improve in overall performance because, regardless of QoS, our algorithms place in cache datasets which are more

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

likely to receive hits. For this reason, the improvement in the performance of high priority datasets does not come at the expense of low priority datasets, rather, all datasets profit, high priority just somewhat more than others.

II. Related Work

There is an extensive literature on cache management schemes. Some basic general references include [4], [3], 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, this method is not as flexible and lowers the overall hit ratio. The mechanism which we use is very simple to implement and to the best of our knowledge, has not been suggested before.

The use of flash drives in enterprise storage systems has been considered in [5], [7], [8], [10], [11], [12], [14], [15], [16], [17], [1].

Applications which may profit from the use of the flash drives have been considered in [5], [7], [8], [10], [11], [12], [14], [15], [16], [17].

The configuration of the back-end of a storage system has been considered in [1], [6].

Configurations of enterprise arrays involving a mix of SSD, SCSI and SATA are commercially common these days and accordingly many vendors offer capacity planning and dynamic data reconfiguration tools for such systems, see [9], [21], [18], [20], [19] among others. While the caching and back-end aspects of a storage system with QoS requirements have been considered separately in the open literature, we are not aware of a previous work which integrates efficiently all these aspects.

III. Background And Preliminaries

We provide a very brief general description of the architecture of the type of storage systems which concerns us in this paper. For more information we refer to [13]. The main physical system components include directors, cache memory and secondary storage devices.

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 the Least Recently Used (LRU) algorithm, which will serve as our baseline algorithm.

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 are of a smaller size of 3.75MB and can be viewed as atomic units for storage statistics collections. In the system that we have studied, a Symmetrix VMAX system, an extent contains 7680 blocks whom are of a fixed size of 512 Bytes.

A. Verification of the Placements Procedure and 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, [2], 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.

B. 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.

C. Cache

In this particular work the cache is divided into 64KB sections and consists of two parts. The major part of it is dedicated to the read requests and the smaller part to write requests. The cache management policies, that will be presented in the next section, refer to the read portion of the cache. The insertion to the write portion of the cache is managed using the LRU policy. The write cache is used for hiding the latency of writing to the disk drive and are destaged asynchronously as described in detail in [13].

IV. Cache Management Policies with the back-end Placement Algorithm

Cache management policies in this paper address the dynamic nature of our caching priority mechanism, which will be explained later in this section. This mechanism is influenced by the system's back-end priority mechanism, whose definition

and analysis were described in [13], and which served us to determine the back-end placement of the data.

The data is split into three priority groups, high, medium and low. The QoS requirements are the desired response times for these groups, which are set to 2, 6 and 12 milliseconds respectively. The management policies, that we are about to present, emphasize the importance of the hit ratio in cache placement location decisions. We also give weight to the priority of the data in order to provide a sufficient response to the QoS requirements. We use the LRU management policy as a baseline for our results. In addition, we take the QoS requirements into account when placing data on the disk drives using the algorithms presented in [13].

Throughout this paper we will be working with *Extents* as our basic storage units. Note that this is much larger than the cache page size that is 64KB in our evaluations. This implies that statistics used for the cache management will be gathered over *Extents*. Extents are large enough that statistical patterns of usage are meaningful, but not too big so as to be coarse grained.

A. The Hit Ratio LRU (HR LRU) Policy

The HR LRU policy serves as the basis for our Priority LRU policy. The main purpose of the HR LRU policy is to prevent cache under-utilization, therefore, if the cache hit ratio of an extent is low then we would like to avoid leaving it inside the cache for a long time.

The policy inserts the extents with a high hit ratio towards the end of the LRU queue, while the extents which are rarely accessed are inserted towards the head of the queue (but not at the very head). Apart from the placement/insertion phase, the cache operates as an LRU queue.

When we consider the cache size in our calculations, we refer to the size of the read part of the cache. The cache time slice is a period that spans 5 minutes. The placement calculation works as follows:

Let $MaxHR_{current}$ be the highest hit ratio among all the extents, whose data was added to the cache or accessed while in it during the current cache time slice.

Let $MaxHR_{previous}$ be the same as $MaxHR_{current}$, but only for the previous cache time slice.

$MaxHR \leftarrow$ max($MaxHR_{current}$, $MaxHR_{previous}$) (in percentage points).

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

$InsertionIndex \leftarrow HR/MaxHR$.

The $InsertionIndex$ is a value which assesses data's hit ratio relatively to the highest relevant hit ratio.

We discretize the $InsertionIndex$ into a small number of intervals, covering the range of all its possible values, i.e. [0,1]. Let there be m such intervals. We will denote interval bounds by x_i, s.t. $1 \leq i \leq m$. Let y_i be a value determining the location in which data will enter the cache ($1 \leq i \leq m$ holds). The y_i is measured in percentage points of the cache size. We will initialize these parameters during our experimental evaluation analysis, which is described later in this section.

Let y_i calculation be as follows:

if $InsertionIndex \in [x_i, x_{i+1})$ **then**

$InsertionIndex \leftarrow y_i$

end if

We refer to $[x_i, x_{i+1}]$ if $i = m$ in the *if* condition.

Final position $\leftarrow InsertionIndex$.

For example, if the $InsertionIndex$=0.45 and belongs to an interval [0.4,0.7) then it will be inserted at 70% of the cache/queue.

The lowest value of y_i, y_1 in our case, will be greater than 0. So, even if some data has very low hit ratio, it will not be placed at the very head of the queue. We can also see that our placement calculation contains hit ratio comparison according to the latest time window, so that the placement of data in cache is more accurate and only the most relevant data's hit ratio is taken into account while computing the placement position.

B. Priority LRU Policy

Let RT be the current response time of an *Extent* and DRT its desired response time, i.e. the current response time and the desired response time of the LU it is a part of. Our goal is for each *Extent's RT/DRT* value to be close to 1, which means to satisfy the QoS requirements, which are not addressed in the HR LRU policy. We do not want it to be way below 1 though, because we wish to utilize our resources and improve the performance of a data, which QoS requirements are not met yet. We have decided to use an integrated priority mechanism, which will handle the QoS requirements throughout the whole system. For the back-end, the priority mechanism stays the same as in [13], which imposes the use of dynamic priorities for the cache as well.

The priority of the data is the one of the *Extent* it resides in. We denote this priority by $prio$ ($minPriority \leq prio \leq maxPriority$), while $minPriority$ and $maxPriority$ values prevent the data from moving too far to either end of the queue. The $prio$ value is measured in percentage points and is altered every cache time slice depending on the RT/DRT ratio of the LU, which the *Extent* is a part of. In addition, we introduce to the system a factor, which we denote by $prioFactor$ (s.t. $|prioFactor|$ is a constant) and which is added to the current $prio$ value every cache time slice. This factor is positive if the RT/DRT ratio is larger than 1 and is negative otherwise. The addition is done only if the new priority value is still legal, i.e. satisfies $minPriority \leq prio \leq maxPriority$.

The cache placement algorithm is based on the HR LRU policy, with a small adjustment added to the final position calculation according to the HR stats of the data. We would like to move the data towards the tail or head of the queue if it has high or low priority respectively, so that the final position of the data also reflects the current damage its LU has suffered based on the comparison to the desired response time. The calculation procedure is as follows:

Let $InsertionIndex$ hold the value assigned to it at the end of the HR LRU Policy placement algorithm.

Let $MinPosition$ be the smallest position in the cache that data may be placed in. This value is the minimum value that $InsertionIndex$ may hold after the completion of the HR LRU Policy placement algorithm, i.e. if the corresponding $HR/MaxHR$ value belongs to the first interval.

$InsertionIndex \leftarrow InsertionIndex + prio$

$PrioPosition \leftarrow min(InsertionIndex, 100\%)$

Final position $\leftarrow max(MinPosition, PrioPosition)$

As we can observe, the final position is a combination of the $InsertionIndex$ value, which was presented in the previous subsection and is defined by the data's hit ratio and priority. In addition there are some sanity checks, which include the verification of the final position being above the minimum position allowed and not beyond the whole cache size.

It is not too difficult to efficiently implement this placement algorithm. As a part of our implementation we have used pointers to approximate LRU queue positions(x-% of the cache) in order to achieve an $O(1)$ access time to the queue.

C. Algorithm Flow

The general time units for our simulation are of the 1 hour length. The simulation basis and most of the actions performed by our system during this time unit are described thoroughly in [13]. There are, however, some additional enhancements that had to be done while simulating the cache activity. Our general time units are further divided into cache time slices, which last 5 minutes. For each time unit, we collect hit ratio statistics per extent. According to this hit ratio data we adjust the maximum hit ratio values for the last couple of time slices. These values are needed for the calculation procedure described in the HR LRU Policy subsection. In addition, we maintain extent's priority value based on total response time and total number of I/O's statistics, which are collected for the LU it belongs to during each time unit, and the desired response time we have assigned to this LU's priority group at the start of the simulation.

D. Experimental Evaluation

In this section we describe the experiments conducted to evaluate the changes, which we have made to our placement calculations for both the HR LRU and the Priority policies in general, and the dynamic priority mechanism for the cache that is introduced in this paper in particular. Our experiments are designed to examine:

- The improvement in each priority group's response time.
- The overall improvement of the response time in the system.

E. Comparison

We have divided all the LUs into three priority groups, each with its predefined desired response time. We ran two simulations, all taking the QoS considerations into account in the back-end part of the storage system. The first simulation ran as a benchmark for our Priority LRU cache management policy and the cache management in it was done according to the regular LRU policy. The second simulation ran with Priority LRU policy. Our goal is to examine the impact of taking the QoS considerations into account in the cache during the simulation with the Priority LRU policy in comparison to the one ran with the plain LRU policy, which does not take the QoS requirements in cache into consideration.

We want to inspect the performance of each simulation by collecting the relevant statistics, such as response times per each priority group, during each general time unit.

F. Analysis

We had data traces from 4 customers. The detailed analysis we present is from one customer over a 24 hour period. Qualitatively similar results were obtained from the other customers as well, and their summary will be shown later in this section. This customer trace contains 98 LUs, with each being of a size that ranges from a few GB to a hundreds of GB. As the traces are from the real customers, the activity along the various time slices is not uniformly distributed. Hence, the response times and other statistics collected by us differ greatly between different time periods. The desired response times were set to 2ms, 6ms and 12ms for the high, medium and low priority group respectively. The simulations used a cache of size 100 GB, 1 SSD (flash) device, 6 FC devices, and 100 SATA disks. Regarding the back-end placement algorithm, we have allowed a maximum of 15000 exchanges of *Extents* between the different devices during each general time unit. These exchanges were distributed according to the system's workload, which in this case happened to be around an average of 600 seconds at the beginning of each general time unit.

We want to analyze system's performance after it has stabilized. Thus, we have removed the first two general time units statistics from our analysis.

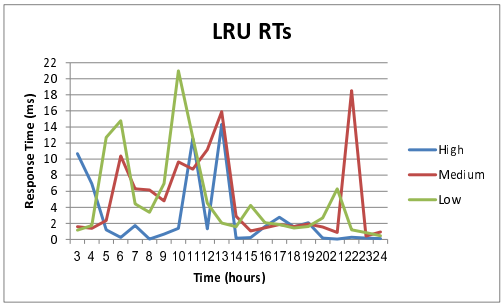
We have predefined 4 intervals and the corresponding 4 y_i values, introduced during the HR LRU Policy placement algorithm description. The intervals are: [0,0.2), [0.2,0.4), [0.4,0.7), [0.7,1] and the values: 20%, 40%, 70%, 100%. We wish to remind that y_i values are measured in cache size percentage points.

As we can see, y_i values coincide with the interval upper bounds values. We have decided to take a conservative approach in setting these values.

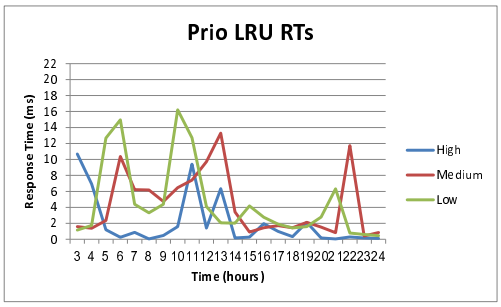
We have fixed the $|prioFactor|$ value to be 5%, as we do not want to move data significantly either to the head of the queue or to its tail at once, but rather do so gradually during each general time unit, while altering data's priority after each cache time slice.

The $minPriority$ and the $maxPriority$ values were set to -30% and 30% respectively. We do not want to hand an excessive weight to the priority in final position calculation, because then we are in danger of leaving some data, that is being accessed rarely, in the cache for too long. On the other hand we want data, whose RT/DRT value is high, to improve its response time.

Both the $|prioFactor|$, $minPriority$ and $maxPriority$ values are measured in percentage points of the cache size.



(a) LRU Response Times



(b) Prio LRU Response Times

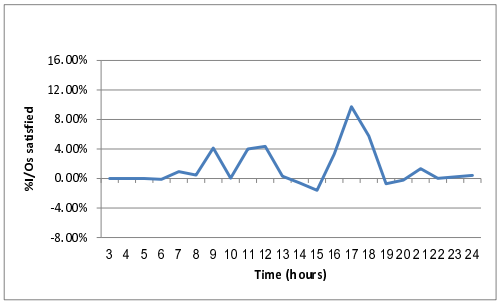
Fig. 1. This figure shows the response times comparison between the LRU and Prio LRU cache management policies.

The response times throughout the simulations conducted on the customer are shown in figure 1. Figure 1(a) shows the response times achieved by the LRU management policy, i.e. response times observed by the end of the simulation, that does not take QoS requirements into consideration at the cache level. Figure 1(b) shows the response times achieved by the end of the other simulation, which ran with our Priority LRU policy and thus did take the QoS requirements into consideration for the cache management. We can observe that overall our Priority LRU policy outperforms LRU policy. For example we would like to highlight the two time slices that are the most problematic in terms of response times achieved by the high priority group LUs, these are the hours 11 and 13. During these times the LRU policy achieves an average response times of 12.55 and 14.32 milliseconds respectively. Our Priority LRU policy reduces the damage to the high priority group data and achieves response times of 9.4 and 6.34 milliseconds for these time slices. From inspecting the figure 2(a) we can denote that the improvement gained during hour 11 can be partially referred to an increase of 4% in number of I/Os satisfied in cache for the Priority LRU policy over the LRU policy. As for hour 13, where there is no such gain in number of cache satisfied I/Os, the improvement can be addressed to the nature of data kept in cache by Priority LRU policy. LRU policy does not try to keep in cache the data that suffered a substantial damage, i.e. has a high *RT/DRT* value, while the Priority LRU policy prioritizes such data. From this we can conclude that the Priority LRU's ability to leave in cache data that suffered more damage enables us to gain an improvement in response times even during periods, when the policy does not hold an advantage in cache hits in comparison to the LRU policy. Overall hour 13 can be emphasized, as the one where our policy is able to substantially decrease the response times of the LRU policy for the high and the medium priority groups without damaging the performance of the low priority group. During this hour the Priority LRU achieves response times of 6.34, 13.29 and 2.06 milliseconds, while the LRU policy achieves times of 14.32, 15.9 and 2.05 milliseconds for the high, medium and low priority groups. Another interesting observation occurs during the times period consisting of hours 17 and 18. From inspecting the figure 2(a) once more we can see that there is a significant increase of 9.74% and 5.76% in cache satisfied I/Os respectively achieved during the simulation ran with the Priority LRU policy in comparison to the one ran with the LRU policy. Therefore, the performance of the Priority LRU policy response times wise is better than that of the LRU policy. During these hours our Priority LRU obtains response times of 1ms and 0.34ms, while the LRU policy only reaches the times of 2.77ms and 1.58ms. The difference in time during hour 17 is more significant as here the Priority LRU is able to stay below the desired response time mark for the high priority group of 2ms while the LRU policy is not.

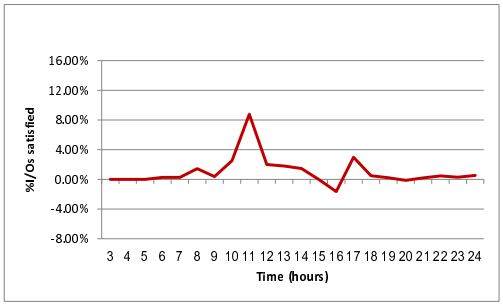
For the medium priority group, we can notice that our Priority LRU policy is able to reduce the damage suffered by this group's data, especially during a particularly harmful periods where the response times are well above the desired response time mark of 6 milliseconds. We can see this pattern occurring throughout the whole simulation. For instance, during a time window, which stretches from hour 10 to hour 13 inclusively, we can observe that the response times of 9.65, 8.75, 11.18 and 15.9 milliseconds achieved by the LRU policy are bettered by our policy, with it achieving the times of 6.48, 7.4, 9.72 and 13.29ms. We can back this observation after looking at figure 2(b), where during this time period we see a consistent increase of around 2 to 9 percent in number of I/Os being satisfied in cache with our Priority LRU policy over the LRU policy. We can observe a similar behavior during hour 22, where our Priority LRU policy reduces the LRU policy's response time of 18.52ms to a less damaging 11.72ms.

As for the low priority group, even in this case the Priority LRU policy manages to perform better than the LRU policy. We can observe some significant damage reduction, e.g. we can see that during hour 10 the response time achieved by our Priority LRU policy is 16.22ms in comparison to a response time of 20.98ms achieved by the LRU policy. This improvement can be addressed to our cache priority mechanism, which increases the priorities of some low priority group's data after making its damage assessments. We can further strengthen this notion by looking at figure 2(c), where there is a substantial increase of 15.17% in number of low group I/Os satisfied in cache with Priority LRU policy in comparison to the corresponding number with the LRU policy.

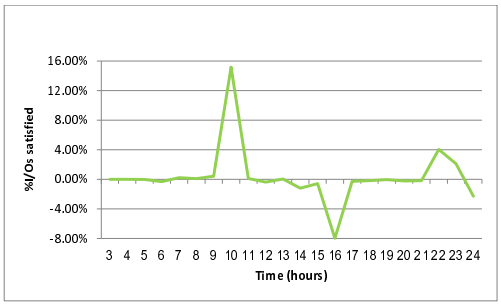
To conclude our analysis of this particular customer we want to emphasize the fact that our Priority LRU policy achieves



(a) High Priority Group



(b) Medium Priority Group



(c) Low Priority Group

Fig. 2. This figure shows, for each priority group, the improvements gained by Priority LRU policy over the LRU policy regarding the number of I/O requests that were fulfilled in cache.

a significant improvements in both the high and medium priority groups' response times without compromising the performance of the low priority group LUs. On the contrary, it manages to outperform the LRU policy even in this aspect.

In table I we can observe the performance summary of both the Priority LRU and the LRU policies for multiple customers. As we can see, we have achieved a substantial improvements in high priority group response times for all customers. In addition, we can also notice the effect our Prio LRU policy has had on meeting the QoS requirements for the customers 1 and 2(the customer we have analyzed in detail), i.e. obtaining response times better than the desired ones for all priority groups, while the LRU policy has failed to do so. This achievement was due to the improvements made in the high and medium priority groups' response times.

We can also conclude, by looking at the summary columns

Customer		LRU	Priority LRU	% Improvement
Customer 1	High	2.36	1.38	71.47%
	Medium	6.35	3.26	94.58%
	Low	2.76	2.04	35.38%
	Total	4.06	2.38	70.56%
Customer 2	High	2.59	2	29.39%
	Medium	6.65	5.57	19.45%
	Low	6.77	6.35	6.57%
	Total	5.08	4.32	17.63%
Customer 3	High	1.75	1.6	9.01%
	Medium	41.31	41.34	-0.06%
	Low	38.5	38.43	0.17%
	Total	35.08	35.04	0.11%
Customer 4	High	3.09	2.49	24.10%
	Medium	11.1	10.95	1.33%
	Low	10.04	10.02	0.18%
	Total	9.59	9.44	1.64%

TABLE I
THIS TABLE SHOWS THE SUMMARY OF RESPONSE TIMES (IN MILLISECONDS) OVER ALL CUSTOMERS, COMPARING PRIO LRU AND LRU POLICIES.

for the third customer, that our Prio LRU policy does not always improve the overall system performance of the LRU policy. In this case it happened because of the sheer number of distinct I/Os in almost every general time unit throughout the simulations, making the devices' queues very large.

As a result of all these inspections, we can infer that Prio LRU cache management policy achieves better results than the widely-used LRU policy in both meeting the QoS requirements and overall system performance.

V. BENEFITS OF A COMPLETELY QOS-OPTIMIZED SYSTEM OVER THE REGULAR MECHANISM

In this section we will compare between a completely optimized mechanism that we have presented in section IV and the regular one, that is not QoS aware, presented as a baseline for comparison in [13].

A. Experimental Evaluation

Here we would like to describe the experiments conducted to evaluate the benefits of introducing the integrated priority mechanism into the storage system. Our experiments are designed to examine the same parameters as in section IV, which are as follows:

- The improvement in each priority group's response time.
- The overall improvement of the response time in the system.

B. Comparison

As in section IV, we divided all the LUs into three priority groups, each with its predefined desired response time. We are comparing between two simulations. The first simulation took the QoS considerations into account in both the back-end part of the storage system and the cache, using the Prio LRU cache management policy for this matter (the same as the second simulation as in the previous section). The second simulation ran without considering any QoS requirements whatsoever,

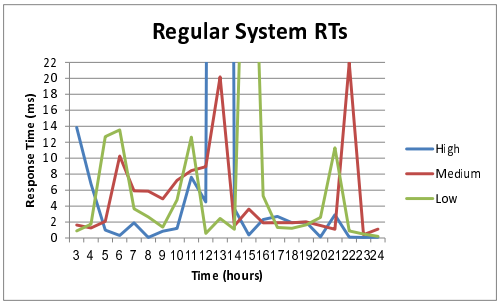


Fig. 3. This figure summarizes the response times for each priority group during the regular system simulation.

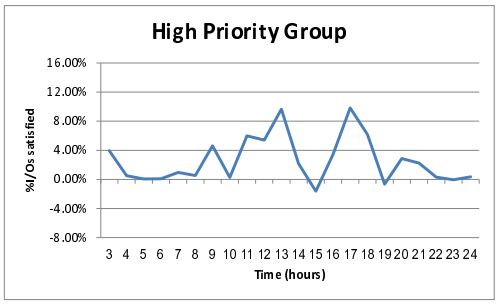


Fig. 4. This figure shows the improvements gained by our integrated priority mechanism over the regular one regarding the number of I/O requests that were fulfilled in cache for the high priority group.

using a non-prioritized mechanism for the secondary storage management and a plain LRU policy for the cache.

Our intent is to show a far greater improvements than those achieved in section IV.

We are collecting the same kind of statistics as described in the previous section.

C. Analysis

As mentioned before we ran our simulations over 4 real customers trace data.

We would like to keep our emphasis on the same customer as in section IV. We have chosen to do so in order underline the continuous improvement from the previous section and thus, build on our mechanism's achievements made so far in this paper. Hence, the system configurations and the simulation parameters remain the same as in section IV.

The response times of the simulation ran on the non-prioritized system are shown in figure 3 and the results of our QoS-optimized system are the same as inspected during the previous section and are shown in figure 1(b).

We will focus our analysis on the high priority group as here there are the most dramatic differences between the two mechanisms and in addition this group is of a great interest to us QoS wise. In order to analyze the differences in high priority group LUs performances we will concentrate on a couple of time periods. The first one is the hours 12 through to 14, where the simulation on the regular system shows the most damage. We can see that the main reason for that is a phenomenally high response time of 414.03ms during hour 13, though the hours 12 and 14 produce response times of 4.53 and 3.7 milliseconds respectively, which is still above the *DRT* mark of 2ms. Our integrated priority mechanism significantly reduces this damage and achieves response times of 1.4, 6.34 and 0.18 milliseconds. This is a combined result of our front-end and back-end optimizations. We can infer that from figures 4 and 5, where there is an increase of around 2.5 to 10 percent in number of I/Os satisfied in cache and a substantial rise of around 41 to 76 percent of I/Os serviced by the SSD devices during this time period achieved by our priority mechanism. An additional time window that we would like to take a look at is the one which spreads from hour 16 to hour 18. There is not so much damage suffered by the high priority group data, but still we notice some important improvements that enable us to meet the QoS requirements. The regular mechanism achieves response times of 2.33, 2.72 and 1.94 milliseconds, while simulation on QoS-optimized system shows response times of 1.96, 1 and 0.34 milliseconds. Thus, our mechanism keeps the high priority group response times under the *DRT* mark of 2ms throughout the whole period. Again, we can address these improvements to a better cache utilization, as we manage to increase the number of I/Os satisfied in cache by some 3.5 to 10 percent and to keeping more high priority data in SSD drives, as we are able to service 18.5 to 31 percent more I/O requests there. We can observe a similar result during the hour 21 as well, where we improve the average response time of 2.93ms to only some 0.05ms.

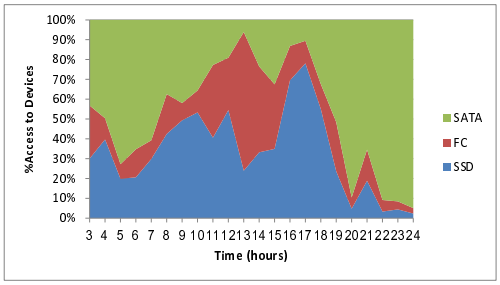
Despite all the QoS implied preferences we have given to the high priority data both in cache placements and in back-end devices assignments, because of its low desired response time, our integrated priority mechanism is still able to utilize our remaining resources in a better way as we improve the performance of the medium and low priority groups' LUs.

For the medium priority group we can see that the performances are quite similar during both simulations apart from two significant time slices. These are hours 13 and 22, where the regular system sustains a substantial damage and records response times of 20.18 and 21.96 milliseconds respectively. The QoS-optimized system statistics show us a decrease in these response times, achieving times of 13.29 and 11.72 milliseconds.

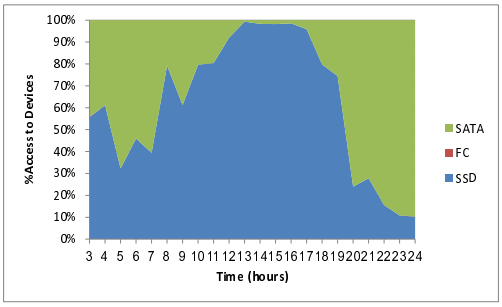
As for the low priority group, we would like to emphasize the hour 15, where we can notice a response time of some 63.26ms achieved during the simulation ran on the regular system. Our priority mechanism manages to better it, however, and record an average response time of 4.19ms during this hour, which is lower the low priority group's *DRT* of 12ms.

Table II summarizes the improvements we obtain by applying our integrated priority mechanism on all the customers.

In this summary we are able to see that the response times of the customers 1,2 and 4 are bettered significantly, with total systems response times' improvements ranging from 28.68 to 263.57 percent. In addition, we are able to reduce the average response times of the high priority group for all the customers



(a) Regular system



(b) QoS-optimized system

Fig. 5. This figure shows the distribution of I/O requests between different back-end storage devices for the high priority group achieved in both the regular and the QoS-optimized systems simulations.

Customer		Reg. System	QoS Opt. Sys.	% Improvement
Customer 1	High	2.29	1.38	66.19%
	Medium	6.12	3.26	87.42%
	Low	2.56	2.04	25.68%
	Total	3.87	2.38	62.70%
Customer 2	High	26.22	2	1210.39%
	Medium	6.77	5.57	21.60%
	Low	13.64	6.35	114.67%
	Total	15.71	4.32	263.57%
Customer 3	High	1.75	1.6	9.33%
	Medium	41.34	41.34	0.01%
	Low	38.42	38.43	-0.02%
	Total	35.06	35.04	0.05%
Customer 4	High	4.37	2.49	75.11%
	Medium	14.35	10.95	31.07%
	Low	12.22	10.02	21.95%
	Total	12.14	9.44	28.68%

TABLE II
THIS TABLE SHOWS THE SUMMARY OF RESPONSE TIMES (IN MILLISECONDS) OVER ALL CUSTOMERS, COMPARING QOS-OPTIMIZED SYSTEM AND REGULAR SYSTEM PERFORMANCES.

by 9.33 to 1210.39 percent.

As for the third customer, there are still no major improvements all due to the reasons explained in the previous section. Additionally, an interesting observation can be made from inspecting the first customer's statistics. If it were for the back-end optimization alone, we would have been unable to achieve any sort of improvement for this customer. We can infer that from comparing an appropriate part of the summary table from the previous section to the regular system's response times. But when we combine this optimization with the front-end one presented in this paper, we are able to achieve improvements for every single priority group.

In conclusion, our examinations of the latter summary table serve as an evidence to the importance of introducing an integrated priority mechanism. Our QoS-optimized system not only enhances the improvements made by our back-end placement algorithm, but also achieves new ones, previously unreachable.

REFERENCES

[1] 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.

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

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

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

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

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

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

[8] I. Koltsidas and S. Viglas. Flashing up the storage layer. In Proc. International Conference on Very Large Data Bases (VLDB), pages 514525, Auckland, New Zealand, August 2008.

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

[10] S. Lee and B. Moon. Design of flash-based DBMS: An in-page logging approach. In Proc. of SIGMOD07, 2007.

[11] 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 10751086, Vancouver, BC, June 2008.

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

[13] G. Lipetz, E. Hazan, A. Natanzon, E. Bachmat, Automated tiering in a QoS environment using sparse data, Proceedings of HPCC13, Zhangjiajie, China, November 2013.

[14] 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 9599, Elmau/Oberbayern, Germany, May 2001.

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

[16] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating enterprise storage to SSDs: analysis of tradeoffs. In Proc. of EuroSys09, 2009.

[17] 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.

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

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

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

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