Automated tiering in a QoS environment using coarse data

Gal Lipetz Etai Hazan Assaf Natanzon Eitan Bachmat

***Abstract*—We present a method for providing automated performance management for a storage array with quality of service demands. Our method dynamically configures application data to heterogeneous storage devices, taking into account response time targets for the different applications. The optimization process uses widely available coarse counter statistics. We use analytical modeling of the storage system components and a gradient descent mechanism to minimize the target function. We verify the effectiveness of our optimization procedure using traces from real production systems, and show that our method improves the performance of the more significant applications, and can do so using only the coarse statistics. Our results show substantial improvements for preferred applications over an optimization procedure which does not take QoS requirements into account. We also present evidence of the complex behavior of real production storage systems, which has to be taken into account in the optimization process.**

I. Introduction

Most of the currently available enterprise storage arrays involve a mixture of SSD, SCSI and SATA drives. Accordingly many vendors offer capacity planning and dynamic data reconfiguration tools for such systems, see [6], [10], [7], [9], [8] among others. Details about the methods of operation of these tools are not publicly available. In addition, the tools do not generally support user defined QoS requirements. Since trace data is not available for large enterprise systems, a realistic algorithm must rely on coarse, counter based statistics which only provide a rough description of the user access pattern.

In this paper we present a dynamic data reconfiguration tool for enterprise level storage arrays. The tool does take user QoS requirements into account and tries to dynamically minimize a damage function which measures the difference between the desired performance of the user applications and the actual performance that the applications experience. We tested the tool on traces from large real production systems. The traces were used to generate the coarse statistics which served as the input to the configuration engine. The configuration engine uses a priority vector which assigns a priority value to each logical unit of data. The priority value and the user access pattern determine together a data placement. The full trace coupled with a rather detailed simulation of an enterprise storage system that we developed, is used to determine application response times. The application response times are then compared to the desired response times resulting in a damage function value. A gradient descent algorithm is then applied to the damage function domain to yield a new priority vector and the process repeats.

Our basic approach which uses widely available coarse data and conservative models is based on an approach which was developed for the SymOptimizer, an external dynamic re-configuration tool for enterprise arrays from EMC. A description of the tool which has been commercially available until recently (superceded by EMC FAST, [6], which operates in a similar mode), since 1998 is provided in [1] with the theoretical background provided in [2].

Our experiments show that our process yields good results on traces from real production systems and substantially improves upon placement algorithms which do not take QoS requirements into account. This is especially significant given the coarsness of the input data. It is unlikely that any external optimization tool will get more refined information, so this study shows that such optimization tools can still be useful. While considering the traces, we observed interesting and complex phenomenon that showed the importance of cleverly handling spikes of activity and in particular of adding device utilization considerations into the placement process. It seems that previous studies have not taken device queueing sufficiently into consideration.

II. Storage System Characteristics

We provide a brief general description of the architecture of the type of storage systems which concern us in this paper.

A. Components

The system is comprised of two main types of components, directors and storage components. The storage components are further divided into primary storage (cache) and secondary storage (disk drives and flash drives).

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. In addition, data can

Gal Lipetz, Department of Computer science, Ben-Gurion University, Beer-Sheva, Israel, 84105. lipetzg@cs.bgu.ac.il

Etai Hazan, Department of Computer science, Ben-Gurion University, Beer-Sheva, Israel,84105. etai@cs.bgu.ac.il

Assaf Natanzon, EMC Corp.,Tel Aviv, Israel and department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel, 84105. assaf.natanzon@emc.com

Eitan Bachmat, Department of Computer science, Ben-Gurion University, Beer-Sheva, Israel, 84105. ebachmat@cs.bgu.ac.il

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 cache comprises a very small portion of the total storage capacity of the system, in the range of 0.1 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 necessary 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 data necessary 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*

There are three types of secondary storage devices, typically used in enterprise storage systems which include SSD-solid state devices (flash), FC devices which are typically 10K or 15K RPM spindles and SATA devices which are typically 5400 or 7200 RPM.

Table I shows the estimated device characteristics that we use to calculate the performance of a particular configuration. The table is based on information from commercial system vendors, but is obviously subject to change due to technological advances.

The I/O overhead is conservatively estimated from server-grade drive data sheets. The overhead is the average latency to complete an I/O, and depends on the drive firmware, interconnect, and rotational latency (for FC and SATA). Sequential I/O is less costly than random I/O in rotating disks, but still incurs some seek and latency overhead especially at the beginning and end of the sequence. We assess no seek or latency penalty for sequential I/O accesses. SSDs do not have a seek penalty, so the latency remains unchanged regardless of whether an I/O is sequential or random.

The read/write rate is the speed at which bytes can be read from the disk. For rotating disks, this speed is relatively stable and relatively close to the speed of flash drives (as compared to the difference in 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 drives as being tiered, with the SSD drives forming the top tier, the FC drives are the middle tier and the SATA drives forming the bottom tier. Our goal is to map the user data to these different tiers in a way that will take into account the user defined performance goals.

III. 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.

Typically, an I/O request will consist of an operation, read or write, a logical volume number, a block offset within the logical volume, both of which identify an extent, and a size in blocks. Using this information, statistics regarding user activity at the extent level are being produced by the storage system and can be used for managing the system. 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. Extents are used for tiering the LU on separate types of drives. Each extent may be stored on a different tier and move between tiers. Our algorithms will make tiering decisions at the extent level, using the extent activity statistics. Specifically, we use the following counters which record statistics for each extent in each LU:

- Read misses
- Read hits
- Writes de-staged
- Number of Bytes read
- Number of Bytes written
- Sequential read requests

We are particularly interested in read misses and de-staged writes, since these actually incur a cost from the storage system drives. The other reads and writes are absorbed by the DRAM cache. The system is also able to identify requests which are sequential and these are reported separately.

TABLE I
ESTIMATED STORAGE CHARACTERISTICS

Type	Capacity (GB)	Cost (Dollars)	Overhead per sequential I/O (ms)	Overhead per random I/O (ms)	R/W rate (MB/second)	Energy (watts)
SSD	300	1000	0.025	0.1	160/120	6
FC	300	100	1	4	60/60	12
SATA	2000	100	2.5	10	50/50	12

The counters summarize the above statistics over a given time period, which could be a few minutes or an hour. The systems that we have studied in this paper report extent activity at 1 hour intervals.

IV. THE PLACEMENT SYSTEM

A. Quality of service

In order to take into consideration the quality of service requirements, we created a model in which the LUs are divided into three groups. Each group represents the importance of the LUs that belong to it. We index the groups by a value which reflects their priority.

Each priority group also has a desired response time DRT associated with it, which determines the desired response time for all of the group's LUs. The LUs that belong to a higher priority group will have a lower desired response time. The placement of the LU's data may change during the simulation, according to the placement algorithm, but its priority group and hence its desired response time will not change.

B. Damage Function

The damage, or score function is used to measure how far away the response time of a volume is from its desired response time. Let $RT_j(t)$ denote volume j's response time during time interval t, and DRT_j denote volume (LU) j's desired response time according to the user's QoS requirements. The damage for each LU is calculated as follows:

if $RT_j(t) \leq DRT_j$ **then**
  $damage_j \leftarrow 0$
else
  $damage_j \leftarrow \frac{(RT_j(t)-DRT_j)^2}{DRT_j}$
end if

The square term in the formula serves as a deterent against large discrepancies between the actual and desired response times. The total damage function is

$$\sum_j damage_j$$

C. The placement algorithm

The placement algorithm is responsible for deciding in which device to place every extent with the goal of minimizing the total damage function.

The challenge is that the algorithm is only allowed to use coarse aggregated data, which consists of the counter values described above. This data is widely available from many customers' storage machines. It takes the quality of service response time goals into consideration.

The placement system's algorithms dynamically decides in which storage tier (SSD, FC, SATA) to place each extent. The change in placements according to the algorithm's decisions occurs every hour. The algorithm uses a priority value $priority_{j,t}$ which is attached to each LU at any given time t and an estimate on the expected amount of activity for each extent, which is denoted by $Act(k,t)$. The priority depends on the QoS requirements of LU j and how far we are from fulfilling these requirements. The algorithm sorts the extents according to $Act(k,t) * Priority_{j,t}$ in order to choose the placement for each extent for the next time unit. The algorithm inserts the extents in the order they were sorted at to the devices until they are full or have reached a maximal estimated utilization allowed, which is 30 percent in our case. It first starts filling the SSD devices, next the FC devices, and lastly the SATA devices.

The calculation of the estimated extent activity is simple. Let $CurrentAct(k,t)$ denote the amount of read miss requests that extent k had during the past time slice. The estimate $Act(k,t)$ will be geometrically weighted to reflect all the historical activity of the extent with an emphasis on the recent past. We compute it as follows

$$Act(k,t) = Act(k,t-1) * g + CurrentAct_k(t) \quad (1)$$

where g is a geometric series coefficient (around 0.95). After calculating each LU's response time, the prediction algorithm performs the priority optimization algorithm described in section **??** which updates the priority of each LU according to its priority group, and the response time seen in the previous time unit.

The computation of the $priority_{j,t}$ is more complicated. Given a new time slice t we perform a gradient descent procedure on the priorities to optimize the expected damage function, given the expected activity estimates. The computation of the expected damage function uses an analytical model of the storage system which will be presented after the gradient descent algorithm. The gradient descent procedure is iterative and can be performed a fixed number of times or until the expected improvement in the damage function is negligible. Below we provide the procedure with a fixed number of iterations which we denote by $Iternum$.

Each LU is given an initial priority of $\frac{1}{DRT}$, where DRT is the desired response time for the LU.

The gradient descent algorithm works as follows:
Let $Priority_j$ denote LU j's current priority.

for $i = 1 \rightarrow Iternum$ **do**
  Sort and place extents
  Calculate expected device utilization and LU response times ERT_j using an analytical model of the storage

system, the estimated extent activities and the current set of priorities

$newDamage \leftarrow$ Calculate the system's estimated damage for the next time slice

if $newDamage \leq minDamage$ **then**

$minDamage \leftarrow newDamage$

$minPriorities \leftarrow LUs' current priorities$

end if

for each LU j **do**

$ratio \leftarrow \frac{ERT_j}{DRT_j}$

$Priority_j \leftarrow Priority_j * ratio$

end for

end for

for each LU j **do**

$Priority_j \leftarrow minPriorities_j$

end for

The algorithm sorts and places extents in the different devices according to $Act(k,t) * Priority_{j,t}$, where extent k belongs to LU j. It then calculates the utilization for each device type and response time for each LU j, denoted by ERT_j, based on the placements chosen in the previous step and the observed activity of the latest time frame. Next, it calculates the total damage of the system (described in section IV-B) resulting from the response times received. If the new damage calculated is smaller than the smallest damage observed so far, it saves the LUs' priorities that resulted in this damage. It then updates each LU's priority by considering whether the LU has achieved it's desired response time goal DRT_j. If the desired response time has not been achieved then the priority is raised. If it has been achieved, the priority is lowered to provide the other LUs with the better resources. The process then continues on to the next iteration. At the end the priorities chosen for the following time unit are the priorities which resulted in the lowest damage.

To apply the gradient descent, after every time interval in which extent statistics are reported, every hour in our case, we need to estimate each LU's estimated average response time ERT_j for the next time slice, based on the current priorities and the activity of the previous time slice.

In order to estimate each LU's response time in a given device we used an M/M/1 queueing model with priorities, [17]. This assumes that in a queue to a device with extents from the three different priority groups, the extents of LUs from the higher priority group are serviced first, and the extents of LUs from the medium priority group are serviced second etc. Some systems do not have the capability to prioritize device queues. In such cases a regular M/M/1 model can be used instead. To apply the M/M/1 with priorities formula we need to calculate device utilizations, respecting LU priorities during the past time interval. We are interested only in the response time of reads since writes are acknowledged immediately once written to cache. Note that the writes will affect the reads by creating backlogs at the devices. This will be taken into account in our model.

Let i denote an index to a device type, say $i = 1$ for SSD devices, $i = 2$ for FC devices and $i = 3$ for SATA devices. We will assume that data is striped in a uniformly random manner over all the devices of a given type and hence that the average utilization is roughly equal for all devices of the same type during a given time interval t. Let j be the index of some LU and let $l(j)$ denote the priority level (group) of LU j. For a device of type i and priority level l, we let $U_{i,l}(t)$ be the average utilization of activity relating to data of priority l and above, during time interval t. If l_{min} is the lowest priority level then $U_i(t) = U_{i,l_max}(t)$ will be the average utilization of a device of type i during time interval t. The utilizations $U_{i,l}(t)$ are calculated as follows: Let n_i be the number of devices of type i. let $D(t)$ denote the duration of the time interval t in seconds. We note that in the systems we studied $D(t)$ was a constant. Let $D_i(t) = n_i * D(t)$. The value of $D_i(t)$ is the total amount of work time, in seconds, during the time unit t for devices of type i.

Let $MR_{i,l}(t)$ be the total amount of MegaBytes (MB) read and $MW_{i,l}(t)$ be the total amount of MB written to devices of type i by LUs of priority l and above, during time unit t. Then, $\frac{MR_{i,l}(t)}{R_i} + \frac{MW_{i,l}(t)}{W_i}$ is the total amount of time it takes to read and write all the data coming from LUs of priority l or higher, where R_i, W_i are the read and write rates of devices of type i in MB per second, as appear in the device characteristics table.

Let O_i denote the average overhead per I/O on a device of type i, as appears in the table and let $miss_{i,l}(t)$ be the number of I/O requests made to devices of type i (misses in cache) from LUs with priority at least l. The value of $miss_{i,l}(t)$ is calculated from the counter data and the list of extents that are currently placed on devices of type i. Then, $O_i * miss_{i,l}(t)$ is the number of seconds spent on overhead relating to requests from LUs with priority at least l, in devices of type i. Summing with the time required to read and write the data, we obtain that

$$T_{i,l}(t) = (\frac{MR_{i,l}(t)}{R_i} + \frac{MW_{i,l}(t)}{W_i}) + O_i * miss_{i,l}(t)$$

is the time required for all read and write operations coming from LUs with priority at least l on devices of type i during time interval t.

The utilization $U_{i,l}(t)$ is then

$$U_{i,l}(t) = \frac{T_{i,l}(t)}{D_i(t)} \quad (2)$$

For an LU with index j, the number $IO_{i,j}(t)$ of I/O generated by LU j to devices of type i during time interval t is calculated using the configuration table and extent counter statistics. Similarly, $MBRead_{i,j}(t)$ and $MBWrite_{i,j}(t)$ which are the number of MB read (written) by LU j to devices of type i during time interval t are also calculated. Note that only read misses and writes to the secondary devices are counted. We let $E(W)_{i,j}(t)$ be the total estimated amount of work (in seconds) generated by the I/O activity of LU j, directed at devices of type i during time interval t. We have

$$E(W)_{i,j}(t) = IO_{i,j}(t) * Overhead_i + (\frac{MBRead_{i,j}(t)}{ReadRate_i}) \quad (3)$$

We note that $E(W)_{i,j}(t)$ represents the estimated response time in the absence of device queues. We take the device

queues into account (and the writes) using the M/M/1 with priorities formula which gives the total response time for requests from LU j's extents in the different device types as

$$ERT_{i,j}(t) = \frac{U_i(t) * E(W)_{i,j}(t)}{(1 - U_{i,l(j)}(t))(1 - U_{i,l(j)+1}(t))} + E(W)_{i,j}(t) \quad (4)$$

Finally, the estimated average response time for LU j will be:

$$ERT_j(t) = \frac{\sum_i ERT_{i,j}(t)}{\sum_i IO_{i,j}(t)} \quad (5)$$

D. Cost Limited Prediction Algorithm

Moving data between devices carries a performance cost due to the transfer of data from one device to another which adds workload to the devices. To control this cost, we developed the Cost Limited Prediction Algorithm which expands the previous prediction algorithm. It adds the ability to limit the amount of extent exchanges between the different devices after each time unit.

The Cost Limited Prediction Algorithm calculates $Act(k,t)$ for each extent k, uses the gradient descent algorithm to calculate each LU's priority, and sorts the extents at the end of each time unit in the same way the prediction algorithm does. After sorting the extents it places them in the devices also in the same way, but only until it reaches the maximum amount of exchanges allowed. If an extent is placed in the same type of device as it already is in it does not count as an exchange. After reaching the maximum amount of exchanges allowed, the rest of the extents remain at their previous location.

V. Verification of the Placements Procedure

We need to verify and measure the improvement resulting from the placement algorithm. 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, [1], 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 storage components. The cache simulation is needed since we need to know which I/Os eventually reach the secondary storage devices. Device queues were also simulated.

A. Trace data

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.

B. Cache

The cache is divided into 64KB sections, and is composed of two parts, read cache and write cache. For each I/O request, the aligned 64KB section (or sections when the size of the I/O is larger than 64KB) of the requested extent will be inserted into the relevant cache section; a read I/O will be inserted to the read cache, and a write I/O to the write cache. The size of the read and write caches vary dynamically according to the number of outstanding writes, not yet written to secondary devices.

The cache simulation includes two cache management policies, FIFO and LRU. In the read cache the insertion and removal of sections are according to the management policy. In the write cache, the insertion is according to the management policy, however, the removal from the write cache is dynamic. The removal from the write cache is determined based on the rate of incoming write I/Os and the size of the write cache in the previous second. According to these values the 64KB sections chosen according to the management policy are removed from the write cache uniformly over the next second. When removing from the write cache the section can is moved to the read cache.

C. Algorithm Flow

The simulation runs in one hour time units. During each time unit the simulation receives all the traces 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 from the storage component it resides in. While handling the I/O, its response time is calculated. 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 device is added to the response time. This time is composed of different parts. Each device has a queue of requests waiting to be serviced. The new I/O enters the queue, and the time it has to wait in the queue until it is serviced is added to its response time. Then, the time it takes to receive the data from the device is included. This time is influenced by the type of device the data resides in since the type affects the read and write rates and the overhead time as described in section II-A.

During the simulation aggregated statistics which will be used by the Placement System are collected. These statistics are collected per extent and contain how many of the requests for each extent were reads and writes, hits and misses, sequential or random, and the total size of the requests in KB.

At the end of each time unit, the simulation runs the Placement System with the aggregated statistics and receives the recommended placements of all the extents in the different

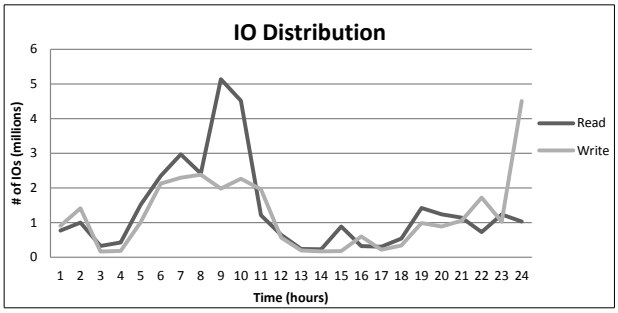

Fig. 1. IO Distribution divided to read IOs and write IOs throughout the simulation

storage components for the next time unit. In order to correctly simulate the exchanges between the different devices, we perform the exchange by reading the data from the device it currently resides in, and writing it to the recommended device. These exchanges add heavy workload to the devices. For instance, in a simulation that allows for 10,000 extent exchanges (each extent is of size 3.75MB, as described in III), we need to move around 37GB of data between the devices. Thus, we distribute the exchanges according to the current system workload and allow up to 50% utilization of the devices, over a time period relative to the number of exchanges to be made. The simulation then continues on to the next time unit as described above.

VI. Experimental Evaluation

In this section we describe the experiments conducted to evaluate the placement algorithm. Our experiments are designed to examine:

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

A. Comparison

All the LUs were divided into three priority groups. We compare our placement algorithm to an alternate algorithm which does not take the Qos requirements into account, by comparing the response time and damage for each priority group. In order to evaluate the difference in each priority group's average response time, we ran two simulations on each data. The first simulation ran without any consideration given to the priority groups. i.e. all LUs were given an equal priority, which stayed static throughout the simulation, only the extent activity was taken into account in the sorting process. The second simulation ran with consideration of the LU's priority groups, adjusting the priorities of each LU as described above.

In order to compare results from the different simulations, during each time unit we calculated statistics per group, such as average response time and total damage, and the total damage of the system. We compared these results to see how the placement algorithm which takes into consideration the QoS requirements influences these statistics.

B. Analysis

We had data from 4 customers. The data we present in detail is from one customer over a 24 hour period. Similar results were obtained from the other traces as well, and will be described later on. This customer trace contains 98 LUs, each of size that varies from few GB to hundreds of GB. The write and read I/O requests are not uniformly distributed, as there are highly active periods, and more relaxed times, as shown in figure 1. The simulation used a cache of size 250 GB, 1 SSD (flash) device, 5 FC devices, and 100 SATA disks, and the LRU cache management policy. We allowed for a maximum of 11000 exchanges of extents between the different devices during each time unit, and these exchanges were distributed according to the workload of the system, which in this case happened to be over an average of 500 seconds at the beginning of each hour.

The results from the simulations for the data described above are shown in figure 2. There are two graphs describing the average response time per priority group. The first graph describes the simulation without any consideration of the priority groups, and the second describes the simulation with our placement algorithm, all described in VI-A. As we can see, there is an overall improvement between the algorithm which does not consider the priority groups, and our placement algorithm. For example, in hour 24, there is an improvement in the average response time of the high priority group, which drops from 4.5 milliseconds, which is above its desired response time, to 1.3 milliseconds, which is below its desired response time. This is due to the higher priority that the LUs in this group have when deciding which extents to place in the SSD devices. This is shown in figure 3 (a) and (b) where all the hits that previously were serviced by the FC drives, and some of the requests that were serviced by the SATA devices, were now serviced by SSD devices. This results in an addition of 400% more accesses to the SSD devices by this group during this hour than in the previous simulation, and an average increase of 80% more accesses to the SSD devices by the high priority group over the 24 hour period. This increase in I/O requests serviced by the SSD reduced these I/Os' response times since the SSD has much better performance characteristics, and thus overall it reduced the high priority group's average response time. During this hour we can also see that the second group's response time increased from 4.3 ms to 5.2 ms, while its cache misses distribution between the devices remained relatively the same, with a small increase in the cache misses serviced by FC instead of the SSD devices. The third group's average response time only very slightly increased in comparison to the no priority simulation, as a few more extents were moved from the SSD to the FC devices. Since the FC devices have worse performance than the SSD devices, and more I/Os were serviced by these devices, the response times for these I/Os increased. This caused the medium and high priority groups' average response times to increase during this hour. After the increase in the medium priority group's response time, it still remained under its desired response time and thus this increase did not hurt the system's overall damage, and it allowed more of the high priority group's requests to be serviced by the SSD devices, and thus lower the high priority group's response time below its desired response time. Not always the decrease in the high priority group's response time comes without a price. As we can see in hour 16, The high priority group's response

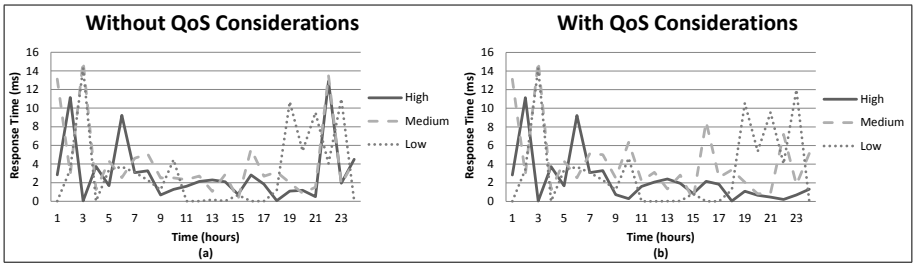

Fig. 2. This figure shows the average response times for each priority group during each time unit. (a) shows the average response times in the first simulation without considering the priority groups, and (b) shows the average response times in our provisioning algorithm

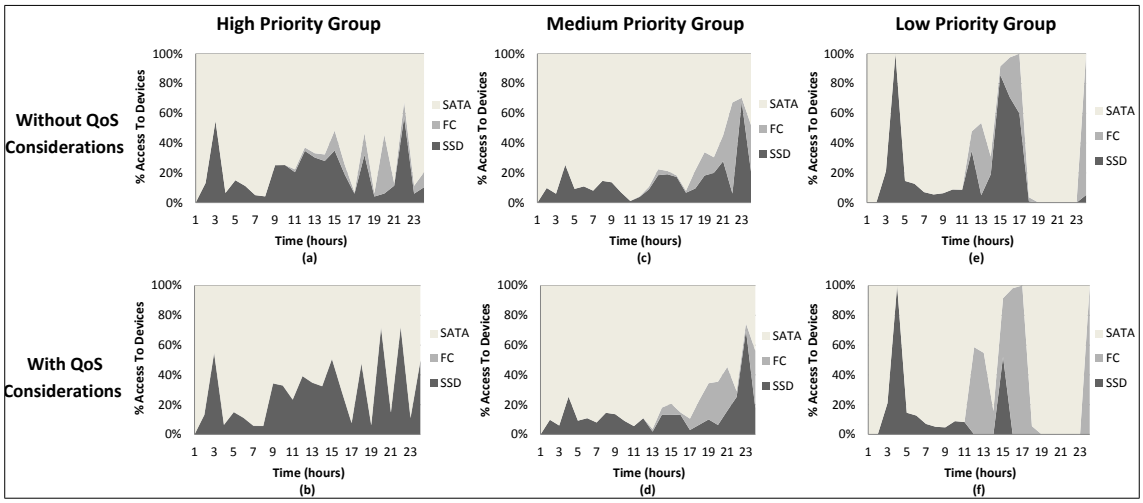

Fig. 3. This figure shows for each priority group, the distribution of IO requests that were fulfilled in each device type

time decreased from 3 ms, to its desired response time, 2 ms. This in turn, caused the medium priority group's response time to increase from 5.6 ms, which is below its desired response time, to 8.5 ms.

We also observed several interesting phenomena. During hour 22, the response time for the high priority and medium priority groups greatly improved, while the low priority group's response time did not suffer. The average response time of the high priority group drops from 13 milliseconds, which is greatly above its desired response time, to 0.2 milliseconds, which is below its desired response time, and the second group's response time also decreased. In the high priority group this is due to the increase in I/O requests serviced by the SSD devices as can be seen in 3 (a) and (b). In the medium priority group the change in the distribution between the devices was slightly different. There was an increase of I/Os serviced by SSD and SATA drives instead of the FC devices. In the low priority group most I/O requests were still serviced by the SATA devices. The improvement in the high priority group's response time without harming the others was caused by a few factors. First, in the high priority group, more I/Os were serviced by the SSD, and since it provides much better performance characteristics, the response times for these I/Os improved. In addition, this decreased the workload on the FC and SATA devices more than the workload that the second and third groups added to it. Thus, the overall workload on the FC and SATA devices decreased, which resulted in a lower queue delay in these devices, which in turn lowered the response time for the I/Os served by these devices as well. This explains why the second group's average response time also greatly decreased, and why the third group's response time stayed the same. These changes resulted in the high and medium priority groups' desired response times being achieved during this hour, and the medium priority group's response time came much closer to its desired than it previously was. A similar behaviour is observed at hour 10 as well. This also demonstrates the importance of queues in storage devices.

Another interesting phenomena happened during hour 15. As can be seen in figure 3 (e) and (f) many I/Os of the low priority group moved from the SSD device to the FC devices, but the average response time of this group only slightly increased. This happened since overall during this hour, for all groups, more I/Os were removed from the FC devices than added to it (when comparing the without QoS and with QoS considerations simulations), and thus the workload on the FC devices decreased, which in turn caused the response times for the I/Os serviced by these devices to be lower than in the simulation without QoS considerations. Even with this reduced workload the performance of the FC devices is not as good as the performance of the SSD devices, and thus the response time for the I/Os of the low priority group did very slightly increase.

During the rest of the hours, there are less substantial

Customer		Total Damage without Priority Consideration	Total Damage with Priority Consideration	% Improvement
Customer 1	High	6.69E+08	1.51E+07	4337.09%
	Medium	3.34E+08	1.50E+08	122.63%
	Low	1.48E+05	2.50E+05	-40.89%
	Total	1.00E+09	1.65E+08	507.28%
Customer 2	High	1.00E+06	8.96E+05	11.75%
	Medium	7.84E+05	7.39E+05	6.09%
	Low	7.88E+04	8.11E+04	-2.80%
	Total	1.86E+06	1.72E+06	8.63%
Customer 3	High	1.95E+12	1.61E+12	20.85%
	Medium	2.06E+11	2.68E+11	-23.41%
	Low	3.99E+09	4.10E+09	-2.75%
	Total	2.16E+12	1.88E+12	14.49%
Customer 4	High	1.05E+14	9.07E+13	16.23%
	Medium	3.64E+13	3.03E+13	20.13%
	Low	2.48E+13	2.09E+13	18.89%
	Total	1.67E+14	1.42E+14	17.46%

Fig. 4. This figure shows the damage from different customers per group and overall for the system

changes in the different groups' response times between the simulations. The main reason this happened is that when we first see an extent which we have not seen before and thus had no previous information about even if it is in the high priority group, it does not receive an advantage over the other extents, and is assumed to have been placed in the SATA devices. However, even with this disadvantage, we constantly see an improvement, even if slight, in the high priority group's response time, and an increase in the low priority group's response time. These improvements bring each priority group closer to its desired response time than in the simulation which did not take the QoS considerations into account.

When observing the damage of the system we also see an improvement with the placement algorithm as opposed to the simulation which did not consider the QoS specifications. During half of the 24 hours of the simulation the damage was significantly reduced. During the first 6 hours the damage stayed relatively the same, as the system had to learn the behaviour and importance of the different LUs, and during only 5 hours it became worse. We can also see that as the system learns the behaviour of the LUs and extents in it, it improves, and thus the damage in the last 6 hours of the simulation constantly and greatly improved. Overall, the damage of the system improved. We can see that when summing the damage throughout the entire simulation, the damage of the high priority group vastly decreased, the damage of the medium priority group slightly decreased, and the damage of the low priority group increased. In addition, and most importantly, the total damage in the simulation with the prediction algorithm was more than 6 times lower than the damage in the simulation without QoS considerations.

We also verified the improvement from our system on other customers, and the results from these simulations are shown in figure 4. We can see that for all customers there was an improvement, for some more significant than others. The percentage of improvement ranges from an 8% improvement to around 500% improvement. Generally, for most customers, we can see that the damage of the high priority group greatly decreases, the damage of the medium priority group slightly decreases, and in turn the damage of the low priority group increases.

VII. Related Work

A comparison of SCSI and SATA drives appears in [5], it should be noted though, that the comparison predates the use of SATA drives in enterprise storage. Various applications which could profit from the enhanced performance of flash drives and the use of flash in enterprise storage systems have been considered in [14], [16], [18], [19], [20], [21], [22], [23], [24], [25].

The configuration of storage systems with disk drives only has been considered in [4]. This work uses traces as input data and is mostly concerned with the configuration of LUs to disks. The analysis is based on a mixture of modeling, extrapolation of device performance tables and bin packing heuristics. Unfortunately, real time traces exist very rarely. In contrast, we use the common LU and extent statistics, avoid the bin packing issues by assuming that data is striped across devices of the same type and use a thin queueing model which does not require traces.

A detailed analysis of performance and power in flash drives is given in [12]. It is shown that depending on the specific workload (reads/writes, sequential/random) both the performance of power consumption of the flash drive may vary. A similar study of power consumption in disk drives, [3] shows that the workload characteristics can also affect the power consumption of disk drives in ways which are similar to the way it affects power consumption in flash drives. The paper [13], provides an analysis of management software and algorithms to avoid write issues in flash drives.

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 [6], [10], [7], [9], [8] among others. Details of the operation of these tools is not publicly available. In addition, to work at such level of detail the tools must be fully integrated with the storage system. at this level of granularity the flash is managed more efficiently with methods which resemble caching.

A recent study, [15], provides a description of a tool which relies on I/O traces to produce the required extent based statistics, which are then used to provide capabilities similar to those of the commercial tools to a prototype of a disk array. The tool is then successfully tested on a single production trace, showing the advantages of mixed configurations as well as dynamic reconfiguration. The modeling in [15] is similar in many ways to our modeling but lacks the queueing theoretic load component, also, given the detailed data, being conservative is not a concern.

REFERENCES

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

[2] 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.

[3] 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

[4] 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.

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

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

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

[8] M. Peters. Netapps solid state hierarchy. ESG White Paper, 2009.

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

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

[11] 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 (WOSP98), pages 199207, Santa Fe, NM, Oct 1998.

[12] 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.

[13] E. Gal and S. Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2): 138163, 2005.

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

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

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

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

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

[25] 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.