Automated tiering in a QoS environment using sparse data

Gal Lipetz [⋆], Etai Hazan [⋆⋆], Assaf Natanzon [⋆⋆⋆], and Eitan Bachmat [†]

No Institute Given

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.

1 Introduction

intro

2 Storage System Characteristics

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

[⋆] Gal Lipetz, Department of Computer science, Ben-Gurion University, Beer-Sheva, Israel, 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

2.1 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 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 sec-

Table 1. Estimated storage characteristics

Type	Capacity (GB)	Cost (Dollars)	Overhead per I/O (seconds)	R/W rate (MB/second)	Energy (watts)
SSD	300	1000	0.0001	160/120	6
FC	300	100	0.0040	60/60	12
SATA	2000	100	0.0100	50/50	12

ondary 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 type 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 1 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 are 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.

The exact energy consumption of a drive depends on its capacity, the number of I/Os it receives, and its idle power consumption. In terms of a single device our table ranks the energy consumption of devices as SSD $<$ SATA $<$ FC. Flash drives use the least energy of any of the drive types, but because they have a significantly lower capacity, the energy consumption of a system comprised of SSDs may be higher than, say, the same system comprised of SATA drives (because it would need many more SSDs than SATA drives to store the same amount of data). This is the case according to the very conservative numbers we have employed in our table.

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.

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.

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 systems 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.

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.

4 System

4.1 Placement System

The placement system is responsible for deciding in which device to place every extent to that its desired response time would most likely be achieved, while taking into consideration all of the Quality of Service requirements.

The placement system uses coarse aggregated data, which consists of the counter values described above. 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, which is denoted by a priority value.

Each priority group also has a desired response time 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.

5 Placement Algorithms

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 prediction algorithm updates the placement of each extent in the different storage devices according to the extent's activity in past time intervals, while taking into consideration the QoS requirements. Each LU is given an initial priority of $\frac{1}{DRT}$, where DRT is the desired response time for the LUN. This priority is dynamic, and can change every hour according to the gradient descent algorithm which will be described in detail in section 5.2.

After every time interval in which extent statistics are reported, every hour in our case, the placement system estimates each LU's average response time. In order to calculate each LU's response time in a given device we used an M/M/1 queueing model with priorities, [21]. 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 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 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 device 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

$$U_{i,l}(t) = \frac{T_{i,l}(t)}{D_i(t)} \tag{1}$$

For an LU with index j, the number $IO_{i,j}(t)$ of I/O generated by LU j to devices devices of type i during time interval t is calculated j 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 are writes to the secondary devices are counted. We let $E(W)_{i,j}(t)$ be the total amount of work (in seconds) generated by the IO activity of LU j, directed at devices of type i during time interval t. We have

$$E(W)_{i,j}(t) = IO_{i,j}(t) * Overhead_i + (\frac{MBRead_{i,j}(t)}{ReadRate_i}) \tag{2}$$

We note that $E(W)_{i,j}(t)$ represents the 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

$$RT_{i,j}(t) = \frac{U_i(t) * E(W)_{i,j}(t)}{(1 - U_{i,l(j)}(t))(1 - U_{i,l(j)+1}(t))} + E(W)_{i,j}(t) \tag{3}$$

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

$$RT_j(t) = \frac{\sum_i RT_{i,j}(t)}{\sum_i IO_{i,j}(t)} \tag{4}$$

After calculating each LU's response time, we perform a gradient descent algorithm described in section 5.2 which updates the priority of each LUN according to its priority group and response time seen in the previous time unit. It then sorts the extents according to $(T\_Activity(extent, t) * Priority(LU of extent)$ in order to chose the placement (the device type) for each extent for the next time unit. *T_Activity* is the extent's total activity from the beginning of the simulation calculated at the end of each time unit by multiplying the activity observed until the previous time unit by a geometric factor g (around 0.95), and then added to the activity in the last time unit - *C_Activity* (read misses, performance measured on reads):

$$T\_Activity(extent, t) = T\_Activity(extent, t-1) * g + C\_activity(t) \tag{5}$$

Priority is the LUN's current priority. After the extents are sorted the system inserts the extents in the order they were sorted at to the SSD devices until the SSD devices are full or reached the maximum utilization allowed. It then inserts the next extents to the FC devices until all the devices are full or the devices reach the maximum utilization allowed, and lastly it inserts the rest of the extents to the SATA devices.

5.1 Cost Limited Prediction Algorithm

This algorithm expands the previous prediction algorithm. It adds the ability to limit the amount of extent exchanges between the different devices after each time unit. Moving data between devices cost in performance due to the necessity to read the data from one device and write it to another, both of which add workload to the drive, and this is done for every extent being transferred. This limit on the number of exchanges can be chosen by the user and is important since and takes a considerate amount of time to perform the transfers.

The Cost Limited Prediction Algorithm calculates the total activity, uses the gradient descent algorithm to calculate each LUN'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 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.

5.2 Gradient Descent

The gradient descent algorithm is responsible for dynamically updating the priority of each LUN at the end of each time unit to one that will satisfy the customer's requirements in the best way possible. The algorithm works as follows:

```
for $i = 1 \rightarrow 10$ do
    Sort and place Extents
    Calculate utilization and response times
    $newDamage \leftarrow$ Calculate system's damage
    if $newDamage \leq Damage$ then
        $Damage \leftarrow newDamage$
    end if
    for each LUN i do
        $ratio \leftarrow \frac{RT_i}{DRT_i}$
        $Priority_i \leftarrow Priority_i * ratio$
    end for
end for
```

The algorithm sorts and places extents in the different devices according to $T\_Activity * Priority$ (similar to what is done in the prediction algorithm). It then calculates the utilization for each device type and response time for each LUN i, RTi based on the placements given in 1. Next, it calculates the total damage of the system (described in section 5.2) resulting from the response times received. If the new damage calculated is smaller than the smallest damage observed so far, it saves the LUNs' priorities that resulted in this damage. It then updates each LUN's priority according to its response time calculated.

Damage Function This function is used to measure how far away the response time of a LUN is from its desired response time. It is used in the gradient descent algorithm to find the total damage of the system after each time unit. For each LUN:

```
if $RTi \leq DRTi$ then
    $damage \leftarrow 0$
else
    $damage \leftarrow \frac{(RT_i - DRT_i)^2}{DRT_i}$
end if
```

where RT_i and DRT_i are the calculated and desired response times for LUN i respectively. The total damage of the system is the sum of the damage of all the LUNs.

5.3 Simulation

The simulation was designed with the storage components and logical units described above. It includes a simulation of the cache and the storage components, with trace data as the input data.

Data This simulation uses real trace data collected from several Symmetrix machines from different customers. This data is hard to collect and requires special permission from the customer in order to collect. Thus, the amount of trace data is limited, which is why the placement system described in section 4.1 does not use trace data but rather it uses aggregated data described later on.

The traces are composed of I/O requests, and each I/O request consists 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.

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 if the size of the I/O is larger than 64KB, of the requested extent will be inserted into the relevant cache; a read I/O will be inserted to the read cache, and a write I/O to the write cache.

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 in the previous second. According to the rate the 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 be moved to the read cache, or removed from the cache completely.

Algorithm Flow The simulation runs in one hour time units. During each time unit the simulation receives all the relevant traces (that occurred during the time unit) 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, and if it is a miss, then it adds the time it takes to fetch the requested section from the relevant device. The time it takes to fetch the data from the device is influenced by a few parameters. 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 it 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 it resides in since it affects the read and write rates and the overhead time as described in section 2.1.

During the simulation aggregated statistics which will be used by the placement system are created. These statistics contain whether the requests were read or write, hit or miss, sequential or random, and the total size of the requests in KB. These statistics will be described in more detail in section **??**.

At the end of each time unit, the simulation runs the placement system with the coarse aggregated statistics and receives back from it the recommended placements of all the extents in the different storage components for the next time unit. The simulation then continues on to the next time unit as described above with the new placements received.

6 Experimental Evaluation

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

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

6.1 Comparison

We compare our provisioning algorithm to an alternate existing algorithm. We compare them using 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. All the LUNs were divided into the three priority groups described in **??**. The first simulation ran without any consideration given to the priority groups. i.e. all LUNs were given an equal priority, which stayed static throughout the simulation, and thus their priority group was ignored. The placement algorithm that ran in this simulation was the same as described in **??**, with the exception of the change in each LUNs priority during the simulation (as the priority remained static). The second simulation ran with consideration of the LUNs' priority groups, using the prediction algorithm described in **??**. To compare results, for each simulation we calculated the average response time per group, the total damage per group, and the total damage for the system during each time unit. We compared the results between the simulations to see how the provisioning algorithm which takes into consideration the QoS requirements influences these statistics.

6.2 Analysis

The data we show here is from one customer over a 24 hour period. It contains 98 volumes, each containing between a few extents and over 1150 extents (an extents is 4MB, so the sizes are between few MB and few GB). As shown in figure 1 the write and read IO requests are not uniformly distributed, and there are highly active periods, and more relaxed times. This simulation used 1 SSD (flash) device, 5 FC devices, and 100 SATA disks.

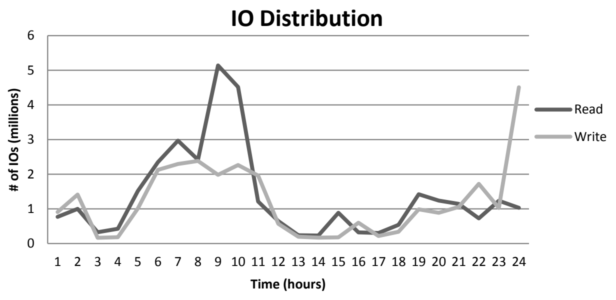

Fig. 1. IO Distribution divided to read IOs and write IOs throughout the simulation.

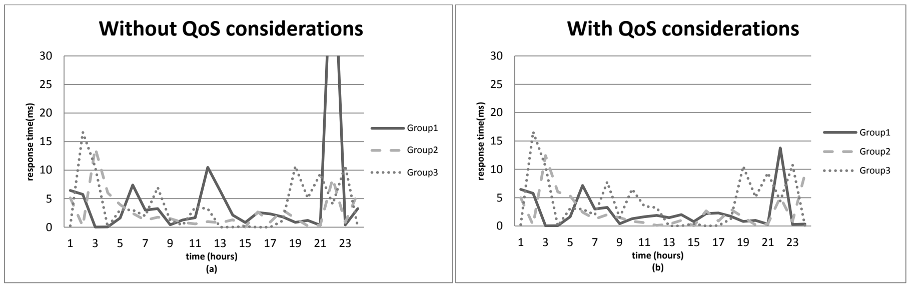

Fig. 2. This figure shows the average response times for each priority group during each time unit. (a) shows the average response times in the first simulation without considering the priority groups, and (b) shows the average response times in our provisioning algorithm.

The results from the simulations for the described above data 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 provisioning algorithm, all described in 6.1. As we can see, there is an overall improvement between the algorithm which does not consider the priority groups, and our provisioning algorithm. For example, in hour 22, there is an improvement in the average response time of the high priority group, which drops from 16 milliseconds which is greatly above its desired response time, to 0.3 milliseconds, which is below its desired response time. This is due to the higher priority the LUNs in this group have when deciding who to place in the flash devices. This is shown in figure 3 (a) and (b) where all the hits that when we did not consider the priority group were fulfilled in the FC, in our algorithm's simulation were fulfilled in the flash, which results in an addition on 50% more accesses to the flash by this group than in the previous simulation. We notice that the amount of hits fulfilled in the SATA stayed constant between the simulations. This is due to the new extents that belong to this group that were first seen during this hour and thus we had no previous information about them which is why they were

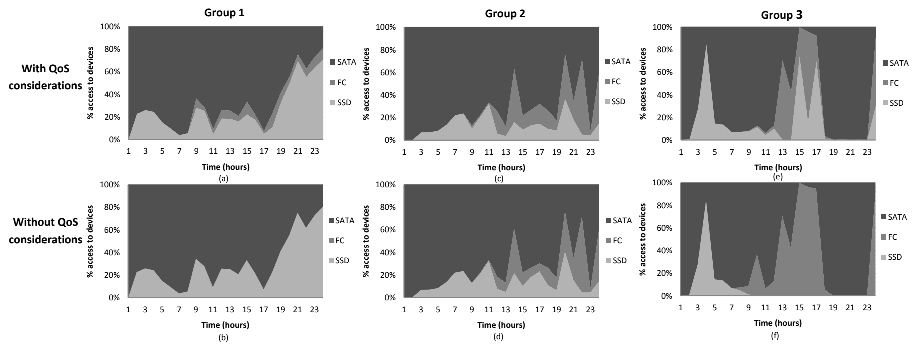

Fig. 3. This figure shows for each priority group, the distribution of IO requests that were fulfilled in each device type.

placed in the SATA devices. This increase in IO requests fulfilled in the flash reduced these IOs' response times since the flash has much better performance characteristics, and thus overall it reduced the high priority group's average response time. In this hour we can also see that the second group's response time increased, while its cache misses distribution between the devices remained relatively the same, with a small increase in the cache misses fulfilled in the FC instead of the flash and SATA devices. The third group's average response time only slightly increased in comparison to the no priority simulation, as a few more extents were moved from the flash to the FC devices. Since the FC devices have worse performance than the flash devices, and more IOs were fulfilled in these devices, the response times for these IOs increased. This caused the medium and high priority groups' average response times to increase during this hour. The medium priority group was increase above its desired response time, but its the price the system pays in order to reduce the high priority group's response time below its desired.

we also observed several interesting phenomena. For example, during hour 24, the response times for the high priority groups greatly improved, while the medium and low priority group's response time barely suffered. In the first group this is due to the increase in IO requests fulfilled in the flash devices. There was an addition of 238% in the IOs that were misses in the cache and fulfilled in the flash in comparison to the number of IOs fulfilled in the flash in the no priority simulation. In the second group the change in the division between the devices was more subtle, with an increase of IOs fulfilled in the FC, and in the third group no IO requests were fulfilled in the flash and were fulfilled in the FC devices instead. The improvement in the high priority group's response time with harming the others was caused by a few factors. First, in the first group, more IOs were fulfilled in the flash, and since it provides much better performance characteristics, the response times for these IOs improved. In addition, this decreased the workload on the FC devices more than the workload the second and third groups added to it. Thus, the overall workload on the FC devices decreased, which resulted in a lower queue delay in these devices, which in turn lowered the response time for the IOs served in these devices as well. This explains why the second group's average response time only slightly increased even though we did not improved the device placement of its extents, and why the third group's response time stayed the same, even though we worsen its LUNs' locations in the storage devices (as more of its IOs were fulfilled in the FC and not in the flash). These changes resulted in all groups' desired response times being fulfilled during this hour. A similar behaviour is observed at hour 12 as well, where the high priority group's average response time decreased by 19 ms, and the medium and low priority group's response time remained unchanged. In most of these cases, our algorithm is able to balance they devices such that the desired response times for each group are achieved.

There were also times when our algorithm worsened some groups' response times above their desired. For example, in hour 16, when the response time of the first group stayed practically the same, but the second group's response time increased dramatically in comparison to the no priority simulation. This occurred as a result of the second group's LUNs receiving less storage in the flash devices and instead being placed in the FC. As we can see in figure 3 (c) and (d) the second group had less accesses to the flash and more accesses to the FC device during hour 16 in the simulation with QoS considerations, which caused its response time to increase when its LUNs are highly active.

During the rest of the hours, there are less substantial changes in the different groups' response times between the simulations. The main reason for that is that when we first see an extent that 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 be 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 our algorithm as opposed to the simulation which did not consider the QoS specifications. During 13 of the 24 hours the damage was significantly reduced in our algorithm versus the no priority considerations algorithm. During 6 hours the damage stayed relatively the same, and during only 5 hours it worsen. We can also see that as the system learns the LUNs and extents in it, and their behavior it learns, and the damage in the last 6 hours of the simulation only improved. Overall, the damage of the system also improved. We can see that when summing the damage throughout the entire simulation, the total damage in the simulation with our algorithm was 30% lower than the damage in the simulation without QoS considerations.

7 Conclusions

8 Future Work

9 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 [18, 20, 22–29].

The configuration of storage systems with disk drives only has been considered in the series of papers, [4],[6],[7],[8],[9],[15] and [31]. 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 [16]. 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 [17], 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 [10, 14, 11, 13, 12] 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, [19], 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 [19] 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.

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 from EMC. A description of the tool which has been commercially available since 1998 is provided in [1] with the theoretical background provided in [2].

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. E. Anderson. Simple table-based modeling of storage devices. Technical Report HPL SSP20014, HP Laboratories, July 2001.
7. E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, Hippodrome: Running rings around storage administration. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST). USENIX, Monterey, CA, 175188, 2002.
8. E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: an approach to solving the workload and device configuration problem. Technical Report HPLSSP20015, HP Laboratories, July 2001.
9. E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang. Quickly finding near-optimal storage designs. ACM Trans. Comput. Syst., 23(4): 337 374, 2005.
10. B. Laliberte. Automate and Optimize a Tiered Storage Environment FAST! ESG White Paper, 2009.
11. M. Peters. Compellent harnessing ssds potential. ESG Storage Systems Brief, 2009.
12. M. Peters. Netapps solid state hierarchy. ESG White Paper, 2009.
13. M. Peters. 3par: Optimizing io service levels. ESG White Paper, 2010.
14. Taneja Group Technology Analysts. The State of the Core Engineering the Enterprise Storage Infrastructure with the IBM DS8000. White Paper, 2010.
15. 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.
16. 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.
17. E. Gal and S. Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2): 138163, 2005.
18. J. Gray and B. Fitzgerald, Flash Disk Opportunity for Server Applications, ACM Queue, Vol. 6, Issue 4, 18-23, 2008.

19. J. Guerra, H. Pucha, J. Glider, W. Belluomini and R. Rangaswami, Cost Effective Storage using Extent Based Dynamic Tiering, Proc. of FAST 2011.
20. S.R. Hetzler, The storage chasm: Implications for the future of HDD and solid state storage. http://www. idema.org/, December 2008.
21. L. Kleinrock, *Queueing Systems. Volumes 1-2*, Wiley-Interscience, 1975.
22. 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.
23. S. Lee and B. Moon. Design of flash-based DBMS: An in-page logging approach. In Proc. of SIGMOD07, 2007.
24. 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.
25. A. Leventhal. Flash storage memory. In Communications of the ACM, volume 51, July 2008.
26. 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.
27. M. Moshayedi and P. Wilkison, Enterprise Flash Storage, ACM Queue, Vol. 6, 32-39, 2008.
28. D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating enterprise storage to SSDs: analysis of tradeoffs. In Proc. of EuroSys09, 2009.
29. 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.
30. M. Wang, A. Ailamaki and C. Faloutsos, Capturing the Spatio-Temporal Behavior of Real Traffic Data, Performance 2002.
31. J. Wilkes, Traveling to Rome: QoS specifications for automated storage system management. In Proceedings of the International Workshop on Quality of Service (Karlsruhe, Germany). Springer, Berlin, Germany, 7591, 2001.