Efficient RAID Disk Scheduling on Smart Disks

Tai-Sheng Chang
tchang@cs.umn.edu
Tel: +1-847-856-8074
Department of Computer Science and Engineering,
University of Minnesota
200 Union Street SE #4-192
Minneapolis MN 55455

David H.C. Du
du@cs.umn.edu
Tel: +1-612-625-2560
Department of Computer Science and Engineering,
University of Minnesota
200 Union Street SE #4-192
Minneapolis MN 55455

1. Introduction

With the emerging high-performance storage systems as well as the availability of faster processors and high-speed networks, many applications that were only dreams a few years ago, have become reality. For example, Digital Libraries and Digital Medical Imaging Archive Systems have become available today. Many of these new applications are making great impacts on the way we work and the way we live. Among the supporting technologies, a high-performance storage system is one of the most critical factors in these systems.

RAID (Redundant Array of Independent Disks) has been playing a very important role in supporting high performance storage systems. It exists in storage systems ranging from one with a couple disks to those with several terabytes capacity. RAID uses data striping and parity information to provide higher I/O throughput on large data access and fault tolerance against disk failure. The implementation of RAID systems can be categorized into two different groups. The first category is the hardware RAID that uses additional RAID controllers to manage and process most of the required tasks in a RAID system. Those tasks include data parity computation and volume management. The other category of RAID uses the existing CPU(s) and memory on the system instead for all the necessary tasks (as opposed to the hardware RAID solution, we call it software RAID). From a user's point of view, hardware RAID solutions require RAID controllers and increase the costs of a system; On the other hand, Software RAID solutions consume CPU and memory resource when performing RAID operations. Therefore, the applications running on the same hosts where the software RAID resides will suffer performance degradation.

Fortunately, there is a new technology that provides an alternative solution between the expensive Hardware RAID solutions and the poorer performing Software RAID solutions. This new technology is called Disk-Based XOR. Disk-Based XOR is a technology utilizing the capability of computation on disks. By calculating the XOR results on disks, the CPU resource is no longer required for the computation-intensive XOR computation in RAID systems. Another big advantage of the Disk-Based XOR approaches is that the data amount needs to be transferred on storage channel can be greatly reduced by as much as 50%. With traditional RAID's, both old data and old parity data have to be sent to the host or a RAID controller for new parity construction. The new data and the new parity will be then transferred back to the target data disk and parity disk, respectively. On the contrary, in a Disk-Based XOR RAID, only the new data and the XOR results of the new and old data will be transferred. Therefore, with Disk-Based XOR, up to twice as many disks could be connected to a storage channel without saturation under the similar load. This advantage has been proved with simulation results in an earlier study.

However, there are challenges in implementing a Disk-Based XOR RAID system. Because XOR calculations of the new and old data will be executed on the data disk and the results need to be transferred to the parity disk, the results have to be saved on data disk before the results have been transferred successfully to the parity disk. It may have a big impact on performance. Researchers have found a potential deadlock situation with traditional single-threaded executions of SCSI commands in Disk-Based XOR RAID's. Some researchers proposed a different RAID parity placement on disks to avoid such a problem. Another research showed the deadlock could be avoided with a small change on the FC-AL protocol. A multi-threaded SCSI command execution approach has been proposed not only to resolve the deadlock problem but also improve disk efficiency. The approach uses a conditionally prioritized disk command queue to resolve the deadlock problem. Simulation results were shown that such an approach outperformed a host-based RAID.

While the proposed multi-threaded XOR approach seems promising, it does raise another issue: The proposed conditionally prioritized disk command queue execution may conflict with disk scheduling discipline designed to optimize disk efficiency. The conflict is due to the fact that free cache segments may not be always available for the next new read-modify-write command. In such a case, one of the other commands will be executed next instead. As a result, a disk may not execute commands as efficiently as it could have been. In this paper, we will investigate the performance impact of such scheduling conflict and propose two new disk scheduling algorithms.

We choose a popular disk scheduling, Shortest Service Time First (or SSTF) as the base line for comparison. This method has been widely used and shown as having good performance in a dynamic environment where commands are arriving over time. In this paper, we call the SSTF scheduling a Greedy Algorithm. In this scheduling, each disk chooses the command with the shortest service time (seek time plus latency time) to be the next command. In the case when available cache segments are not enough for next read-modify-write operation, the command with the shortest service time among the other commands will be chosen. This is the same as in the proposed multi-threaded approach by other researchers in their study. The only difference is that in this paper, SSTF scheduling discipline will be used to choose from the list of executable commands. When no other commands are in the disk queue, a disk will be forced idle.

Two reasons may cause disk cache to build-up. The first is due to congested data links. When the disks are putting data to cache faster than cache can transfer data to the storage channel, the cache will be filled. This could happen when too many disks are connected to a single storage channel. This situation can be easily avoided with proper sizing when configuring a system if the traffic load can be realized. In Disk-Based XOR, there is another possible cause. Disk cache segments filled with XOR results need to be protected until the associated parity update is completed. Depending on the disk scheduling discipline, a parity update command may take a long time waiting in disk queue before it has been executed. The longer the waiting time is, the longer time the associated cache segments on the target disk remains to be saved and protected from being used by other commands. Our proposed approaches will intend to reduce the waiting time of the parity updates.

The rest of this paper is organized as the following. In Section 2, we will provide a more detailed description of Disk-Based XOR operations. In Section 3, we will also describe in details the Greedy disk scheduling discipline and those two new enhancements. In Section 4, we will present our simulation results to show the performance of those three disk scheduling disciplines following an overview of our simulation model. Finally in Section 5, we will summarize what we found in this study and conclude the paper.

2. Disk-Based XOR and Its Operations

Three new SCSI commands (see [1]) have been created for supporting the Disk-Based XOR implementation. They are XD-write (or XDW), XP-write (or XPW), and XD-write extend (or XDW-ext). Each XDW is always associated with one XPW command. An XDW command consists of four operations. To begin, data (old data) will be read from target disk to its disk buffer (disk cache). At the same time, new data will be sending from the host to the target data disk. When both new and old data become available on disk buffer, exclusive-or operations will be executed on the new and old data. The new data will be written onto the disk. The results of the XOR operations, on the other hand, will remain on the disk buffer for later use by the associated XPW. The results need to be saved and protected on the disk buffer from being overwritten by other operations. Figure 1 shows an XDW operation.

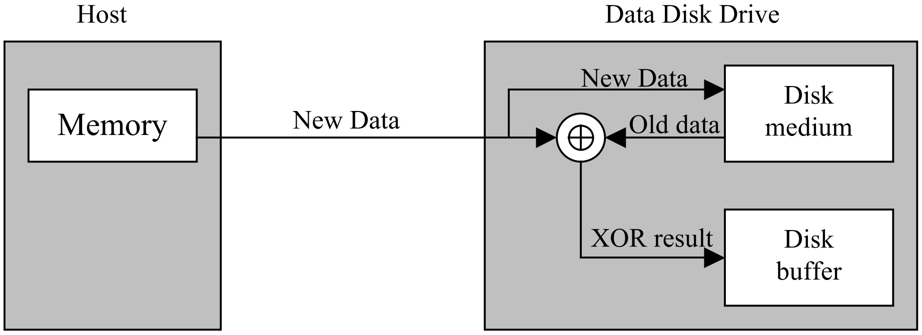

Figure 1: XDW Operation

After an XDW command is completed, the associated XPW command will be sent to the associated parity disk. The old parity will be read from the disk medium. At the same time, the XOR results of the associated XDW command stored earlier on the target data disk will be sent to the parity disk. When the XOR results and old parity information become available, XOR operations will be executed. The newly derived XOR results will be written onto the parity disk. After the XPW has completed, the disk buffer storing the XOR results saved on the target data disk by the associated XDW will be freed. Figure 2 shows the operations of an XPW command.

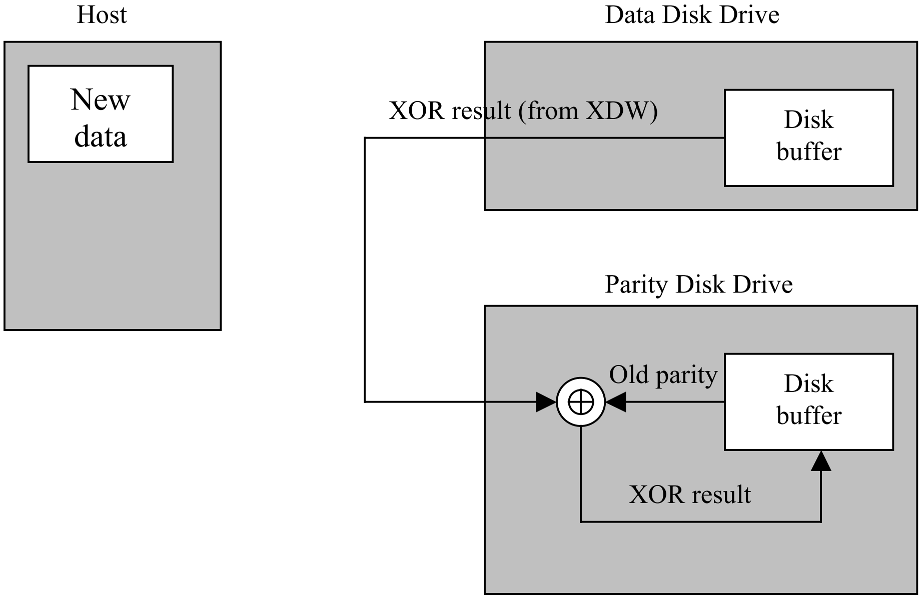

Figure 2: An XPW Operation

An XDW-ext command is a macro command that consists of one or more XDW commands followed by the associated XPW command(s). A read-modify-write operation on a data block can be fulfilled by an XDW-ext command.

One big advantage of the Disk-Based XOR approach is that the data amount being transferred on storage channel can be greatly reduced by as much as 50%. With the traditional RAID's (either hardware or software RAID's), both old data and old parity data have to be firstly sent to the host or a RAID controller to construct the new parity data. The new data and the newly derived parity data will be transferred back to the target data disk and parity disk, respectively. In other words, if we need to update a block of data, there will be four blocks of data that are required to be transferred from and to the disks. As opposed to the traditional RAID's, in a Disk-Based XOR RAID it only needs to transfer the new data and the XOR results of the XDW on the storage channel. Therefore, with Disk-Based XOR, a larger number of disks can be connected to a storage channel before saturating it with the same disk load.

3. Two XPW-Enhanced Disk Scheduling Disciplines

Many disk scheduling disciplines have been proposed to improve disk efficiency. For example, SCAN and C-SCAN ([2]) were proposed to reduce the seek time without moving back and forth from one request to another. Some other approaches considered to reduce both seek time and rotation latency (i.e. disk service time). Shortest Service Time

First (SSTF) is one of those approaches and has been widely used as the disk scheduling discipline.

Before RAID was first introduced, disks operated individually and independently. There was no correlation between any two operations on different disks in terms of their access location on disks. RAID changed such independency. Updating a data block on one disk in a RAID will result in updating the associated parity block that has the same Logical Block Address (LBA) as the data blocks but resides on a different disk (parity disk). However, most disks in a RAID (except RAID-3) are still operating independently without coordination between disks. That is, reading the old data from a disk is performed independently with the reading of the associated old parity data from another disk. Because the new parity data is constructed by the old data, old parity data and the new data, intermediate results must be saved before both the old data and old parity are available. Without collaboration, the retrievals of the old data and old parity will be scheduled independently on two disks. As a result, the intermediate results may have to be saved for a long period of time. That is why most of the RAID systems require a large amount of memory either on the RAID controller or on the host.

Such a big memory requirement is impractical in a Disk-Based XOR RAID. With a very limited buffer space on most disks, disk buffer can be filled quickly with Disk-Based XOR operations. When the disk buffer is full, no more commands will be executed until some buffer becomes available. A more severe condition is that a deadlock may happen when the buffer is full in Disk-Based XOR. That is why in [3], the proposed conditional prioritized disk scheduling forced a disk to choose a command other than XDW-ext after the occupancy of the disk buffer is higher than a predefined threshold. However, such an alternation on the disk scheduling will have an impact on the disk efficiency. The disk efficiency could be much lower when choosing a sub-optimal command.

In the following, we will introduce two XDW-enhanced algorithms. Both of them are intended to reduce the probability of being required to make a dramatic change on disk scheduling. As for a baseline comparison, we use a greedy algorithm with the SSTF scheduling. The discussion of this Greedy scheduling approach is also included in the following sections.

3.1 Greedy Disk Scheduling

The Greedy algorithm chooses the command with the shortest service time (seek time plus latency time) to be the next command to be executed. This method has been widely used and performs well in dynamic environment where commands are arriving over time. We use this method as a baseline for comparison purpose.

Because cache may be filled in Disk-Base XOR as discussed in the previous section, some modification is needed when applying the Greedy method to Disk-Based XOR RAID's. Each XDW-ext command requires at least two segments of cache to store data; one for the old data from disk and another for the new data from the host (assuming request data size is less than or equal to the segment size). Hence, we need at least two segments of free cache space in order to start execution of an XDW-ext command. When the number of available cache segments is small enough for the next XPW-ext command, we change the Greedy Algorithm and choose the command with the shortest service time from commands other than XDW-ext commands. The modified greedy method is used in this paper as a performance baseline to compare with the proposed (two) enhancements.

As discussed in the previous section, one drawback of the Greedy method in Disk-Based XOR is that when it is running out of free cache space, it has to pick a sub-optimal command, or even worse, stay idle. In a case when there is no command other than XDW-ext in the disk queue, the disk has to stay idle until either a new non-XDW-ext command arrives or some cache space is freed.

One straight forward way to reduce such inefficiency is to prevent it from happening. There are two reasons causing the cache to back up. The first is due to a congested link. When the disks are putting data to cache faster than cache can transfer data to the storage channel, the cache will be filled. This could happen when too many disks are connected to a single storage channel. This problem may be eliminated with proper system sizing when configuring a system.

In Disk-Base XOR, there is another possibility. That is when the number of outstanding XDW-ext commands on a disk is close to the number of cache segments. An outstanding XDW-ext command is an XDW-ext command finishing its XDW part but waiting for its XPW part to be complete on another disk. Depending on the disk scheduling discipline, an XPW command may take a long time waiting in disk queue before it is executed. The longer the wait time, the longer the cache segment on the data disk needs to be saved and protected from being used by other commands.

After understanding the cause of a long-waiting outstanding XDW-ext command, we proposed two approaches to reduce the possibility of filled cache in Disk-Based XOR RAID's. The details are in the next two subsections.

3.2 An XPW Service Time Based Promotion Scheme (XPWT)

The first approach is to selectively give an XPW the higher priority. By giving XPW commands higher priority, it helps to reduce its wait time in disk queue and as a result, the associated XDW-ext command can be completed and release the cache space it used earlier. However, selecting XPW should be made with caution such that the disk efficiency will not be over-compromised. We use a relative difference in disk service time as the criteria to give an XPW the higher priority. When an XPW has less than smallest service time plus the predetermined time δ available, the XPW with the smallest service time will be given the highest priority and will be executed next.

We formulate the approach proposed above in the following.

Let C_{All}^{min} be the command with the shortest service time T_{All}^{min}
Let C_{XPW}^{min} be the XPW command with the shortest service time among XPW commands T_{XPW}^{min}.

If $T_{XPW}^{min} - T_{All}^{min} <= \delta$ then choose C_{XPW}^{min} to be the next command.
Otherwise choose C_{All}^{min}.

Note that when δ equal to zero, this approach degenerates to the Greedy Algorithm. On the other hand, when δ becomes a large number, XPW commands will be given the higher priority all the time. For example, when δ is greater than or equal to the largest possible disk service time, the above method will always give the higher priority to XPW commands.

3.3 An XPW Queue Length Based Promotion Scheme (XPWQ)

The performance of the previous approach highly depends on the value of δ. Choosing a large δ may result in lower disk efficiency but reduce the number of XPW's in disk queue; while choosing a small δ makes it closer to the Greedy Algorithm. Therefore, the optimal value of δ is difficult to determine in a dynamic situation. The second approach we are proposing in this paper is to give XPW commands the higher priority when the number of XPW commands on a disk reaches a certain threshold. The idea is based on the fact that with a uniformly distributed access among disks in a RAID and a large number of XPW commands in one disk queue, the more occupied disk cache will be on the other disks. Therefore, choosing an XPW to execute will likely help in releasing the disk cache buffer on another disk. Furthermore, when the threshold is chosen properly, there will be a set of XPW commands in disk queue to choose from when the number of occupied cache segments reaches the threshold. The larger the number of XPW commands to choose from, the closer the chosen XPW command to the optimal command. The detailed formulation of this approach is provided in the following.

Let MaxNxpw be the threshold value of the number of XPW commands.
Let Nxpw be the number of XPW commands in a disk command queue.

If Nxpw <= maxNxpw then follow the Greedy Algorithm.
Otherwise, pick the XPW command with the shortest service time of all XPW's.

Note that when the value of maxNxpw is set to zero, this approach will always choose an XPW if one exists. On the other hand, when the value of maxNxpw is set to infinity, then this approach will not give XPW a special higher priority at any case. Therefore it will degenerate to the Greedy Method.

4. Simulation Model and Results

In this section, we will use simulation results to demonstrate the performance difference of the three disk-scheduling disciplines discussed in the previous section. For better understanding of the simulation results, we first provide an overview of our simulation models in the following subsection.

4.1 Simulation Model

We used a storage subsystem simulation model to simulate operations of a storage subsystem based on the Fibre Channel - Arbitration Loop (FC-AL) ([5]) protocol. The model consists of three major components: A disk and its disk cache component; A storage interface component that follows FC-AL protocol and controls data transfers to/from the storage channel; And a command generator component that simulates a host generating data requests.

4.1.1 Disk and Disk Cache Model

The disk model is based on an IBM Ultrastar XP 4.51GB disk. The implementation of this disk model employs zone bit recording and non-linear seek time functions for read and write operations using information from the disk manufacture in [6]. Table 1 shows a summary of disk parameters used in the simulation.

Table 1: Disk Parameters

Disk Parameters	Value
Capacity	4.51 GB
Rotation Speed	7202.7 RPM
Average rotation latency	4.17 ms
Seek times	0.5 – 16.5 ms
Transfer rate	5.53 – 7.48 MB/sec

Disk cache is the buffer for temporarily storing data sent to/from the storage interface. It is partitioned into segments. Each segment consists of many 512-byte blocks. In our simulation model, each segment will be used by one command. The cache component also employs an LRU (Least Recently Used) cache segment replacement scheme. The parameters that the disk cache used in the model are summarized in Table 2. In our simulation, the number of segments is a controlled parameter. We used different numbers of segments in order to understand the impact of cache size and disk scheduling schemes on the system performance.

Table 2: Disk cache parameters

Disk Cache Parameter	Values
Block Size	512 bytes
Number of segments	Varied
Segment size	64 KB

4.1.2 FC-AL Model

We follow the FC-AL standard to model our disk interface. FC-AL is a protocol allowing Fibre Channel to operate in a loop topology. It is logically located between FC-1 and FC-2. The FC-AL component in our model consists of both Loop Port State Machine (LPSM) and Fibre Channel Protocol for SCSI (FCP). LPSM defines the behavior of the FC-AL loop port. It includes an arbitration protocol which determines who can access the loop. It also includes a fairness protocol that enforces fair sharing of loop among all the nodes. FCP is one of the Fibre Channel mapping protocols (FC-4) which uses the service provided by FC-PH to transmit SCSI commands and data. It also transmits status information between a SCSI initiator and a SCSI target. More details about FC-AL can be found in [5] and [7]. Table 3 summarizes the parameters we used in the FC-AL model.

Table 3: FC-AL Simulation parameters

FC-AL Simulation Parameters	Values	Descriptions
Link Speed	100 MB/Sec	Bandwidth of an FC-AL loop
Propagation Delay	3.5 ns	Propagation delay between two nodes
Per Node delay	6 word time	The delay of forwarding a frame by interface
Fairness algorithm	Enabled	The fairness protocol in its arbitration scheme

4.1.3 Command Generator

Command Generator is responsible for generating commands in our model. At the beginning of each simulation run, it will generate the number of commands indicated by the value of the maximum outstanding command parameter. When a command finishes, it will generate another command immediately to maintain the maximum outstanding commands in the system. The target disk of each command and the command's access location (LBA) on the disk will be randomly assigned by the command generator. The Command generator is also responsible for sending the SCSI command response to the target disk and generating data to be written on disks.

4.2 Simulation Results

To better understand the impact of disk scheduling on Disk-Based XOR, we conducted simulations in many different scenarios. We compared three disk scheduling disciplines under different system loads with different data request sizes. We also compared them in small and large-scale storage systems. To predict the impact of the three different disk scheduling algorithms on the Disk-Based XOR RAID performance with the high-end disks, we further conducted simulations using a disk model with a two times improvement in the disk rotation and seek times. By conducting these different simulations, we hope to provide a better view of the impact of the disk scheduling on Disk-Based XOR RAID performance and therefore, to demonstrate its importance.

To better present our results, we will use an eight-disk FC-AL model as a base model. We will compare the performance by changing the system parameters such as system load, data request size, and number of disks while keeping the other parameters the same. As the base model, We will show the average command response time for 4KB read-modify-write requests in the eight-disk FC-AL system. The total number of outstanding commands was 768. That is, the number of outstanding commands was maintained at 768 after the simulation started. A new command was generated immediately after a previous command had completed. For the XPWT scheduling, the value δ was set to 3 milliseconds. That is, an XPW command was given the higher priority over XDW commands if its disk service time is less than the smallest service time among all the XDW commands plus 3 milli-seconds. The maxNxpw value was set to the number of segments minus four. That is, the XPW commands in disk command queue will be given a higher priority when the total number of XPW commands in that disk command queue is greater than the number of cache segments minus four. For example, if the number of cache segments is twelve and there are more than eight XPW commands in disk queue, the next command will be chosen from those XPW commands in the queue. In such a case, the XPW with the shortest service time among the XPW commands will be chosen as the next command.

The simulation result of the base model is shown in Figure 3. The XPWT Algorithm has the least average command response time among the three on all the cache segment sizes used in this study. It was 7% better than the Greedy algorithm when the number of segments is eight. The results of the XPWQ Algorithm varied with the number of segments. When the number of segments was eight, it performed closely to the XPWT Algorithm. When the number of segments increases, the response time fell between those of the Greedy Algorithm and the XPWT Algorithm.

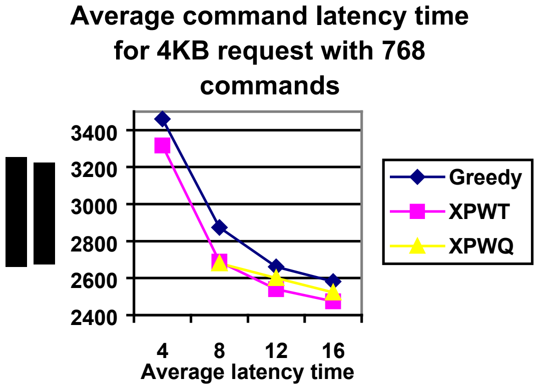

Figure 3:Average command latency with 4KB requests and 768 outstanding commands.

Figure 4 shows the system throughput achieved by the three scheduling algorithms on the base model. Since the system was loaded with a fixed number of outstanding commands (768 commands), the throughput was highly dependent on disk efficiency. The more efficient the disk is, the higher throughput it will generate. In Figure 4, we see that the XPWT Scheduling had the highest throughput among the three methods and had about 7% higher throughput than that of the Greedy Method in certain cases.

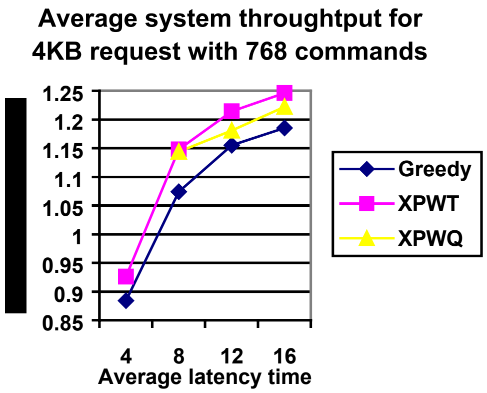

Figure 4:Average system throughput with 4KB requests and 768 outstanding commands.

Different System Loads

To understand the impact of the three different scheduling methods under different levels of the system loads, we also investigated the performance difference with a different number of outstanding commands in the system. As opposed to 768 outstanding commands, we conducted simulations with 512 outstanding commands on the 8-disk model. Figure 5 shows the results with both 768 and 512 outstanding commands. With 512 outstanding commands, the average command latency time was about two thirds of the time with 768 commands. The XPWT method outperformed the other two with 512 outstanding commands in all the three numbers of segments. The difference between the Greedy Method and XPWT Method was reduced from about 7% with 768 outstanding commands to about 5.4% with 512 outstanding commands. From the results, we found that the larger the number of outstanding commands, the higher the performance gap is between the XPWT method and Greedy Method. The major reason is that with more outstanding commands, it is more likely to execute an XDW command than an XPW command. When the cache segments are all filled, the disk will be forced to execute an XPW command. In such a case, the efficiency of the disk will be compromised.

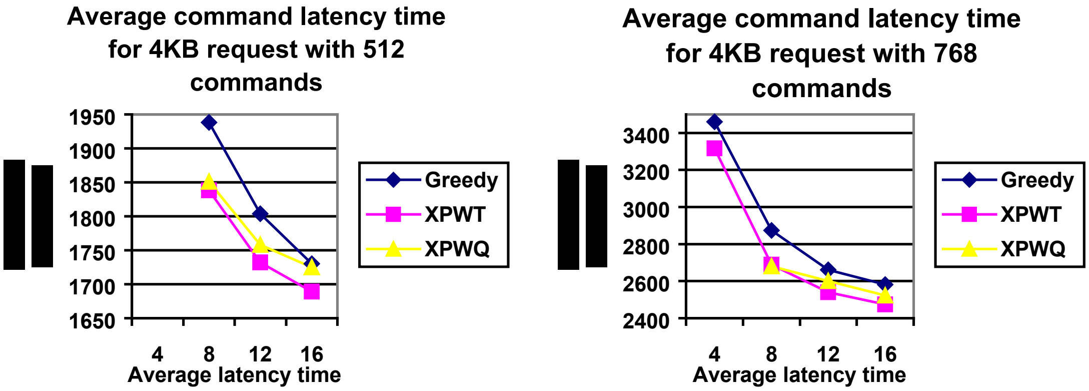

Figure 5: Average command latency with and 512 vs. 768 outstanding commands with 4KB requests.

Another observation from the results is that the XPWQ method tended to be close to the performance of the XPWT Method when the number of segments is small. On the other hand, it tended to be close to the Greedy Method's performance when the number of segments is large. This is because when the number of segments is large, more XPW commands are allowed in a disk queue before they are given the higher priority. Therefore, most of the time, the XPWQ method may perform as the Greedy Method. While with a smaller number of segments, it is more likely to reach the maxNxpw threshold. Therefore, it performs closer to the XPWT Method.

Large Scale Disk System

We conducted simulations on a 32-disk FC-AL model to show the performance in a system with a larger number of disks. In order to eliminate the performance difference resulted from disk queuing time between the eight-disk and 32-disk model, we used the same system load on both systems. We used an average of 64 commands per disk. That is, we used 512 outstanding commands on the eight-disk model and 2048 commands on the 32-disk model. The results showed a similar trend to what we have observed in the eight-disk model (See Figure 6). The XPWT Method was still the best among the three. It is about 7% better than the Greedy Method when the number of segments was equal to eight. The XPWQ Method performed just as well as the XPWT Method when the number of segments was equal to eight. But the XPWT method outperformed the XPWQ method when the number of segments became larger.

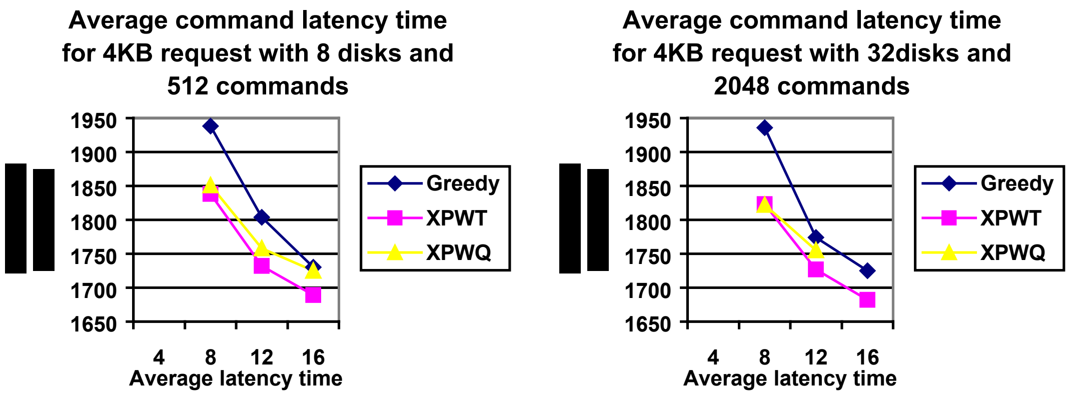

Figure 6: Average command latency with 8 vs. 32 disks with 4KB requests.

Large Request Size - 64KB:

With a 4 KB request size, the actual transfer time is less significant compared to the disk seek time and latency time. Therefore, the disk scheduling has a greater impact on the disk efficiency. As the request size increases, the data transfer time becomes larger. The extent of the improvement with better disk scheduling may be different. To understand the performance of the three disk scheduling disciplines with larger requests, we also conducted simulations with 64 KB requests. The results are shown in Figure 7.

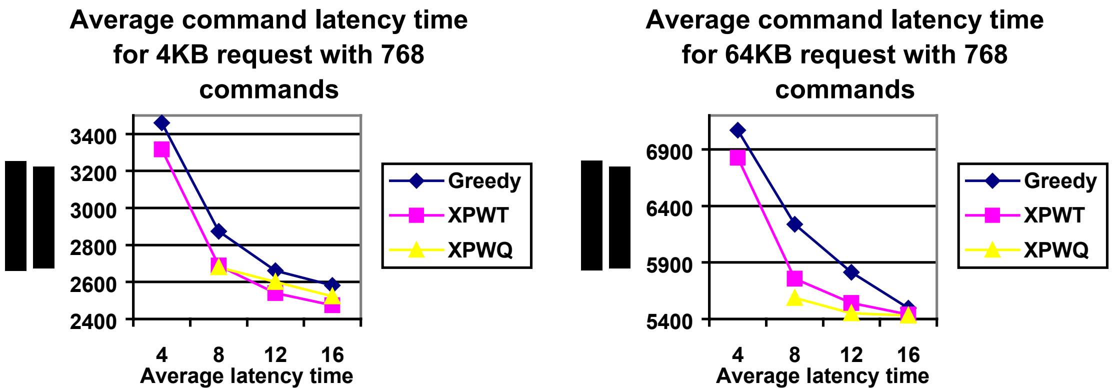

Figure 7: Average command latency with 4KB vs. 64KB.

With 64 KB requests, we observed even better improvement than 4 KB requests with XOR-enhanced scheduling when the number of segments is small. For example, with 4 KB requests, the improvement of the XPWT Method over the Greedy Method was about 7% with 8-segment cache. While with 64 KB requests, the improvement was more than 8%. Furthermore, the XPWQ Method outperformed both the other methods and had an improvement of close to 12% over the Greedy Method with an 8-segment cache.

Performance with the Faster Disks

Disk technologies have improved significantly over the past decades. Recently, disk density has been doubling better than every couple years. The disk rotation speed and seek time have also improved significantly. In this paper, we have compared the performance comparison of different disk scheduling disciplines with disk rotation speed that is used by most of the current off-the-shelf disk products (at the time this paper was written). To predict their performance with the faster disk speed, we also conducted simulation with faster disks.

In order to reuse our disk model and its very detailed seed functions and zone-bit encoding, we modeled the next generation disks by changing the parameters in our existing disk model. With the targeted 15000 RPM next generation disk, we believe that by doubling the disk rotation speed and halving the seek time and data transfer time in the disk model we have, it will give us a close approximation of the model for the next high-end disk. Figure 8 shows the performance comparison of the three scheduling methods with current and high-end disk models. The result is shown in Figure 8. The improvement of the XPWT method is almost 10% better than the Greedy method. The improvement of the XPWQ method fell between the Greedy method and XPWT method. It has about a 6.7% improvement over the Greedy method at eight segments.

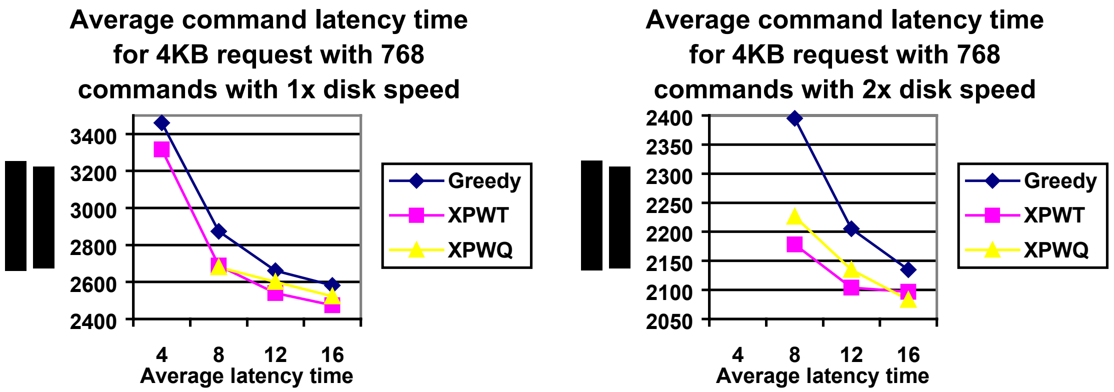

Figure 8: Average command latency with 1x vs. 2x disk speed with 4KB requests.

Impact of δ value in XPWT method

In the earlier section, we mentioned that choosing a good δ in XPWT could be difficult. To understand the impact of δ on the performance, we conducted more simulations with different δ values in different loads and cache segments. Figure 9 shows the results of the average latency when δ changes. The results show that when the number of outstanding commands is 768 and the number of segments is four, we should use a greater δ value. When the number of outstanding commands is 512, the optimal value falls when δ is around three to four. The results also demonstrate that when the number of segments is small, δ should be set to a greater value. In Figure 9, it seems that setting δ to 3 could provide a performance gain close to optimal except when the number of outstanding commands is 768 and the number of cache segments is four.

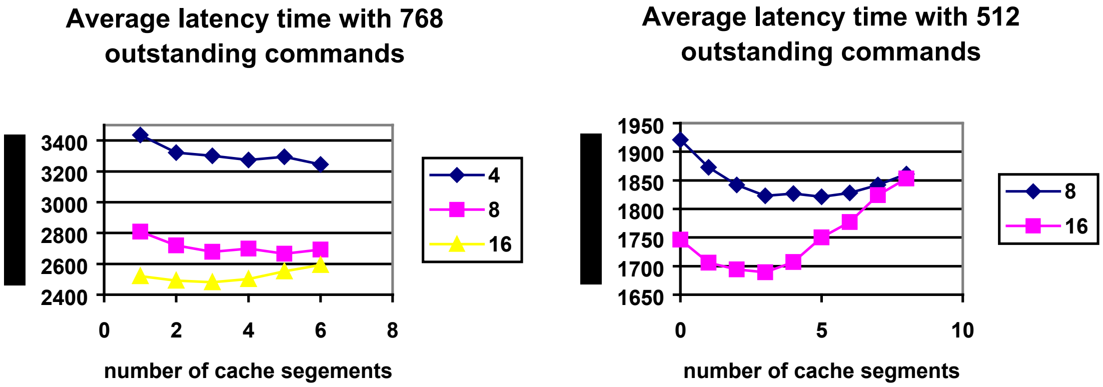

Figure 9: Average command latency with 1x vs. 2x disk speed with 4KB requests.

5. Conclusion

In this paper, we have discussed the uniqueness of Disk-Based XOR operations on disk scheduling and its impact on disk efficiency. We have proposed two XPW-enhanced disk scheduling disciplines that are designed to improve the disk efficiency on Disk-Based

XOR RAID's. We have demonstrated their performance results by simulations. We have investigated the performance of the proposed XPW-enhanced disk scheduling as well as the SSTF approach serving as the baseline performance. We have conducted simulations under different scenarios such as different scales of storage system, different system loads, different request sizes, and even with high-end disk technologies. We have demonstrated using simulation results that the performance was consistently improved with those two XPW-enhanced approaches throughout all the cases. The results showed that the improvement could be as much as 12%.

As the disk technologies continue to improve rapidly, it has been predicted that a one terabyte disk costing below one hundred dollars could be on the market in less than five years. With the price of disk going lower and lower, and the capacity of disks going higher and higher, it becomes more important to have a better RAID solution. Disk-Based XOR provides a promising lower-cost high-performance alternative. We hope that the study we have presented in this paper could open a door to finding better RAID solutions.

References

[1] Gerry Houlder, Jay Elrod, and Mike Miller, "*XOR Commands on SCSI Disk Drives*", X3T10/94-111r9.

[2] Avi Silberschatz and Peter Galvin, "*Operating System Concepts*", Addition-Wesley Publishing Company, Inc. fourth Edition, 1995.

[3] Sangyup Shim, Yuewei Wang, Jenwei Hsieh, Tai-Sheng Chang, and David H.C. Du, "*Efficient Implementation of RAID-5 Using Disk Based Read Modify Writes*" Technical Report, Department of Computer Science, University of Minnesota, 1996.

[4] Tai-Sheng Chang, Sangyup Shim, and David H.C. Du, "*The Designs of RAID with XOR Engines on Disks for Mass Storage Systems*", Sixth NASA Goddard Conference on Mass Storage Systems and Technologies in Cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems, March 22- 24, 1998, College Park, Maryland.}

[5] David H.C. Du, Tai-Sheng Chang, Jenwei Hsieh, Yuewei Wang and Simon Shim. "*Emerging Serial Storage Interfaces: Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (FC-AL)*", TR 96-073, Technical Report, Department of Computer Science, University of Minnesota}

[6] IBM Corporation, "*Functional Specification, Ultrastar XP Models*", 1995.

[7] David H.C. Du, Jenwei Hsieh, Tai-Sheng Chang, Yuewei Wang and Simon Shim, "*Performance Study of Serial Storage Architecture (SSA) and Fibre Channel - Arbitrated Loop (FC-AL)*", to appear in IEEE Concurrency