The Designs of RAID with XOR Engines on Disks for Mass Storage Systems[*]

Tai-Sheng Chang, Sangyup Shim[†], and David H.C. Du
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
Email: {tchang,du}@cs.umn.edu Phone: +1 612 626-7522 Fax: +1 612 625-0572
[†]: Department of Computer, Information, and Systems Engineering,
San Jose State University, San Jose CA 95192-0180
Email: sishim@email.sjsu.edu Phone: +1 408 924-4058

Abstract

Recently, the exclusive-or computation capability was added on high performance disks to provide an alternative way of implementing RAID (Redundant Arrays of Independent Disks). This alternative approach reduces the traffic on the storage channel and eliminates the XOR computation load on the host or RAID controllers. Therefore, it may be a better approach than the traditional RAID design to implement RAIDs in a mass storage arena. In this paper, we propose two new approaches with the XOR capability on disks to implement RAID systems. Simulation results are also provided.

1 Introduction

There have been a variety of RAID (Redundant Arrays of Independent Disks) systems proposed to improve I/O performance. It improves I/O performance by increasing concurrent accesses to disks. By using extra parity information, the RAID systems also provide fault tolerance when a disk fails. Data on a failed disk can be reconstructed with data and parity information from other disks. As a result, it improves data availability. Nonetheless, the update of data (e.g., write operations) requires extra tasks (exclusive-or computations) to update on the parity information.

The traditional RAID implementations rely on a centralized device to perform all the RAID management tasks and the exclusive-or (XOR) computations. Such a centralized device could be either a RAID controller for hardware RAID or the host CPU for software RAID systems. (For convenience, we call this traditional approach as the ”host-based XOR” approach). Since the XOR computation is performed on a centralized device, the old data and old parity have to be transferred over the storage channel to a host or a RAID controller when data updates (*write* operation) or data reconstruction in a case of the disk failure. It results in not only more data traffic on the storage channel, but also higher buffer requirements to store all the temporary data. Also, the XOR computation could be a potential bottleneck with high bandwidth storage channel. In reality, the hardware based

[*]This work is partially supported by Seagate Technology, Inc.

RAID controllers are expensive and can only connect to a relatively limited number of disks. On the other hand, the software RAIDs do not perform as well as the hardware based RAID controllers although they are much cheaper. It is because the XOR computation and other RAID management functions need to be carried out on the host and will compete with all the other applications on the system resources such as CPU, host memory, system bus, and etc.

As an alternative solution to the host-based XOR approach, the XOR computation can potentially be performed on disk drives. (We call this ”disk-based XOR” approach.) Current Seagate Barracuda 9 FC-AL disk drives are indeed capable of supporting XOR SCSI commands proposed in [1]. With this disk-based XOR approach, XOR computation can be performed independently on disks. Also the old data and parity blocks do not have to be transferred to a central device to accomplish the XOR computation. It reduces half of the data traffic on the storage channel for *write* operations. Therefore, disk-based XOR seems to be a better solution for supporting a large scale storage systems compared to the traditional RAID implementations.

In this paper, we investigate the performance of the disk-based XOR approach. We proposed two new approaches with disk-based XOR. The details are described in Section 2. In Section 3, we will show the simulation results for these two approaches. In Section 4, we conclude out study and suggest some future works.

2 Disk-based XOR approaches

The new SCSI command proposed in [1] to accomplish disk-based XOR is called *XD Write Extend* (denoted by XDW-EXT). When a disk (target disk) executes an XDW-EXT, it will read the corresponding old data from the disk. It will then do the exclusive-or computation on this old data and new data (to be sent from host). It keeps the XOR result in the buffer and write the new data to the disk. At the same time, the target disk will send another command called *XP Write* (denoted by XPW) to the associate parity disk. When a parity disk executes such a command, it will read the old parity from the disk. When the parity disk is ready, the XOR result on the target disk will be sent to the parity disk. After the new parity is calculated, it will be written to the parity disk.

There are two implementation alternatives to process an XDW-EXT command. The first is to let the target disk wait for the completion of the operation on the parity disk before it can process the next command. That is, the target disk will not execute the next command in its command queue before the current XDW-EXT is completed. This approach is simpler and needs no other extra function. But it may result in lower disk utilization. The other approach is to allow the target disk to proceed the next command once the XOR result of the new and old data has been calculated and the new data has been written to the disk. This approach allows multiple commands pending on each disk. It would improve the disk utilization but require some mechanism to protect all the temporary data from overwritten.

Since the disks serve commands independently, a deadlock may also happen with such XDW-EXT operations with either approach. This is because it needs two disks to complete its tasks and may cause a deadlock with a circular waiting condition. One major challenge for implementing RAID with the disk-based XOR approaches is to prevent deadlock from happening while maintaining acceptable performance. We propose two different deadlock prevention approaches in this paper. The first approach is to avoid the deadlock condition by re-arranging the location of the parity blocks. For example, when all the parity blocks are stored on one disk (RAID-4), the circular-waiting condition will never happen. Our first approach is based on this observation and the idea of the RAID-5's parity block distribution among a group of disks to improve the performance. We partition the disks into several groups and store all the parity blocks of one group onto another group of disks. We call it "Grouping" approach. For example, if there are n groups, G$_1$, G$_2$, to G$_n$, we may store the parity blocks of group G$_1$ on the disks in group G$_2$. And store the parity blocks of group G$_2$ on the disks in group G$_3$ and so on. In general, we store the parity blocks of group G$_{i-1}$ in group G$_i$ where i is from 1 to $n-1$. Figure 1 shows an example of this approach with three disks in each group (two disks in Group 1). The advantage of this parity placement approach is that there is no need for change on disks. Neither the applications nor the disks require any change. The disadvantage of this approach is that, some disks in the last group G$_n$ will have lower space utilization. To compensate this problem, we can use less number of disks in the last group since the last group is exclusively used for parity block in this approach. The load will be less since there is no additional load from data update on these disks.

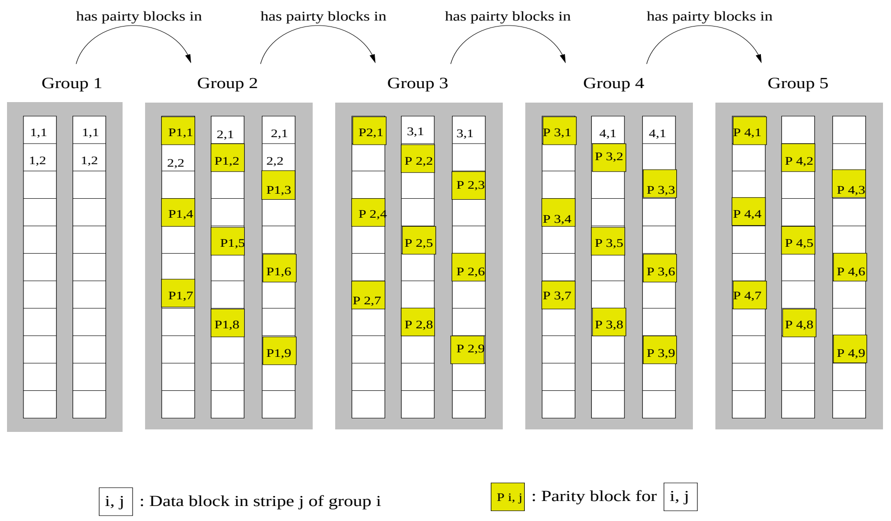

Figure 1: An example of Grouping Approach with 3 disks in each group

In the second approach (we call it the XPWF approach), we resolve deadlock by giving the XPW a higher priority and also ensure that the XPW will be completed once it is sent to the parity disk. The higher priority can be ensured by putting the XPW command at the head of the command queue on the parity disk. To ensure the XPW can be completed, each disk sends the associated XPW first when it starts an XDW-EXT command. After the XPW is sent to the parity disk successfully, the target disk starts processing the associate data update and XOR calculation (We call this portion of XDW-EXT command to be XDW). If an XPW from another disk arrives while a disk is trying to send out an XPW, then the disk with the arriving XPW should abort its current XDW-EXT command and stop the intent to send the associate XPW. The aborted XDW-EXT should be put back to the command queue. Figure 2 shows a control flow of this approach. Since the XPW has not been sent yet, the XDW-EXT has not been processed yet. And the overhead of aborting this command can be minimized since no data has been read and processed at the time of abortion. This XPWF approach works if the transmissions on the storage channel are serialized. That is, only one transmission is allowed at any time. Both SCSI and FC-AL belong to this category. In the case when multiple concurrent transmissions are allowed, such as in SSA (Serial Storage Architecture [3]), some modifications to provide acknowledgments from the parity disk are necessary. The advantage of this approach is that no modification on the applications is required. Also, we can use RAID-5 or whatever parity placement approaches. The disadvantage is that it requires some modification on the protocol of executing an XDW-EXT command. Also this approach is considered pessimistic because it does not necessarily result in a deadlock by sending out an XPW while there is an XPW from other disks waiting for service in its command queue.

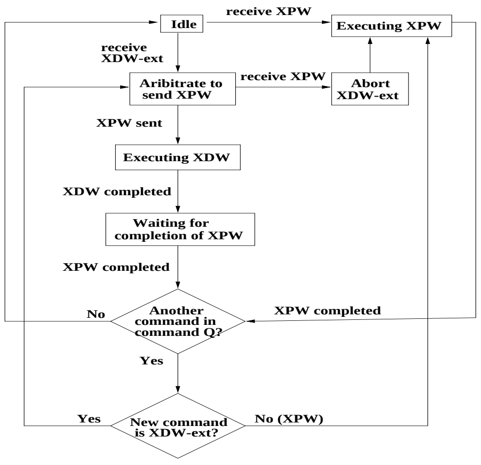

Figure 2: Control flow for XPWF Approach

3 Simulation Results

In this section, we will show the simulation results for the Grouping and XPWF approaches we proposed in previous section. The simulation model is based on the FC-AL model used in [3]. We keep a fixed number of outstanding commands on a host. The number is set to eight times the number of disks attached. That is, each disk has eight commands on average. It represents a case when the disks are highly loaded (each disk is either waiting for XPW to complete or reading/writing data).

The simulation results are shown in Figure 3. The disks were assumed to wait for the XPW's completion before it can process the next command. We also compare the results when multiple outstanding commands are allowed on each disk with the RAID-5 parity block placement. The results show that before the FC-AL link starts to be saturated (≤ 40 disks in Figure 3), both approaches achieved as much as two thirds of the throughput of the method allowing multiple concurrent commands on each disk. This is because in the single command case, the disks have to wait for its parity update before it can process the next command. Therefore, disk utilization is lower. When the number of XPW's are the same as that of XDW-EXT on a disk, the disk utilization will be less than 67%. This is because that each XDW-EXT needs to wait for the completion of its XPW. That is, the disk will be idle at least for the period of time executing an XPW. Assuming data and parity update need the same amount of time on average and the time is one unit, the disks will be idle for at least one time unit (waiting for XPW to complete) for every three units (each XDW-EXT needs 2 units and each XPW needs one unit). That is why the upper bound for the disk utilization is two thirds. When the delay on the loop becomes longer, the disks utilization will be lower.

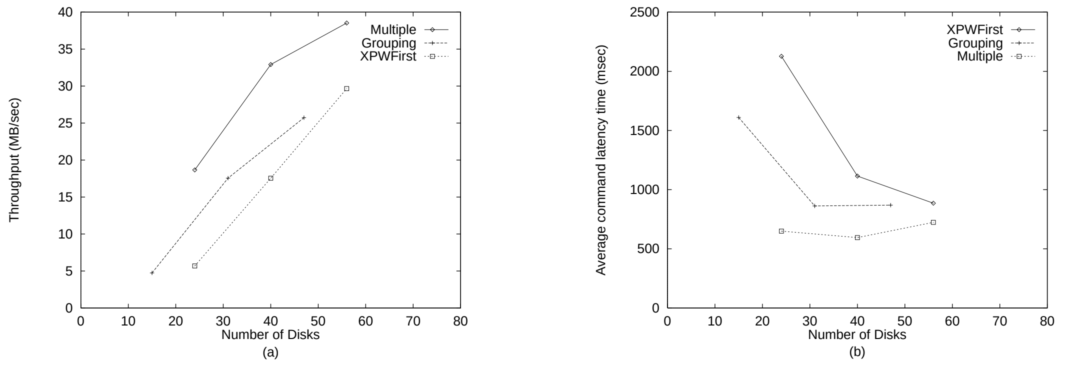

Figure 3: (a): The aggregate throughput comparison; (b): Access latency comparison

Figure 3(b) shows the average access latency for each request. The latency time include the command waiting time in the disk command queue, the disk service time and the data transfer time on the loop.

4 Conclusions

In this paper, we have proposed two different approaches to implement the Disk-based XOR. The simulation results showed that the Grouping approach is slightly better than the XPWF approach. Both approaches achieved about two thirds of the throughput of the multiple outstanding command approach.

In this paper, we have proposed two approaches to implement a RAID system by taking advantage of the XOR capability on disks. Further studies are necessary to investigate its performance with large *writes*, when rebuilding a failed disk, and its buffer requirement.

References

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

[2] 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.

[3] 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 Parallel and Distributed Technology*