# Design and Implementation of a Fibre Channel Network Driver for SAN-Attached RAID Controllers

Jae-Chang Namgoong and Chan-Ik Park
Pohang University of Science and Technology
System Software Laboratory
San 31, Hyoja-dong, Pohang, Kyungbuk, Korea
{sinclair,cipark}@postech.ac.kr

## Abstract

*Fibre Channel SAN is considered to be a promising solution to address storage problems caused by the sheer volume of data and their management. To adopt this new storage environment, we design and implement a high performance Fibre Channel network driver for SAN-attached RAID controllers in a real-time operating system. This paper describes the architecture of the Fibre Channel driver which consists of two modes; a target mode and an initiator mode. An exception handling mechanism for enduring a disk failure is also given. Lastly, we measured the performance of the Fibre Channel driver. Testing results reveal a moderately successful performance of the Fibre Channel driver.*

## 1. Introduction

In today's computing environment, the efficient storage and management of massively growing data has become a major challenge. To solve this problem, a new class of storage solution called storage area networks (SANs) is appeared, mostly depending upon Fibre Channel gigabit networking technology. This storage trend also affects the RAID subsystem architecture to adopt Fibre Channel technology to their internal and external I/O channel interfaces, which are conventionally configured by SCSI channel. Inspired by this technology transition, we design and implement a high performance Fibre Channel network driver for PosRAID[16] developed by the System Software Laboratory at POSTECH. This paper discusses the details of our Fibre Channel driver architecture, and its performance results.

The Fibre Channel driver is divided into two parts; a target mode driver and an initiator mode driver. We introduce the design and operation flow of each driver mode in turn. Next, we outline a scenario to explain how exception events, such as disk failure, can be properly recovered. Furthermore, the performance of the Fibre Channel driver is measured. Finally, we present our conclusions.

## 2. Architecture

PosRAID, a foundation of the Fibre Channel driver, is a RAID subsystem using a PC motherboard as the controller and VxWorks real-time operating system as the platform of the control system. It consists of a RAID operation module for managing RAID level-dependent operations and buffer caches, and an I/O communication module for exchanging messages and data with the host system and its interior disks.

Two QLogic QLA2200 Fibre Channel adapters are installed into the PosRAID system; one for a target mode operation and the other for an initiator mode operation. Therefore, the Fibre Channel driver must support these two operations as well. We shall discuss these in detail.

### 2.1. Target mode operation

The target mode operation communicates with the host system. Namely, the target mode driver must act as a Fibre Channel disk device in order to be accessed as a big logical disk by the host system.

The request from the host system is transferred to the target mode driver accepting an IOCB (I/O control block)[8] through the host memory queue interface from the firmware. Next, the target mode driver parses the IOCB message and passes it over the RAID operation module in the case of a read or write request. When the RAID operation module is ready to transmit data or status, the target mode driver resumes the transaction by issuing a new IOCB to the firmware. Then the firmware transfers the data or status to the host system, or back again. After completing the



0-7695-1153-8/01 $10.00 © 2001 IEEE

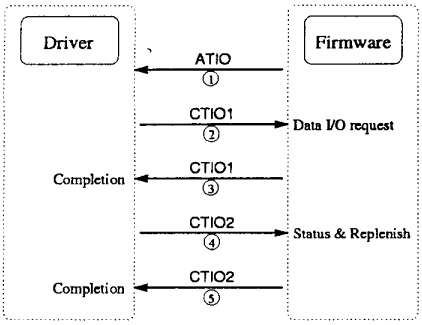



**Figure 1. IOCB message flow in target mode operation**

transmission, the target mode driver releases the resources allocated for the transaction.

Figure 1 summarizes how to exchange the IOCB messages between the target mode driver and the firmware. In read requests, data and status transfer are done at once, so that the exchange of IOCB messages take place three times in total. However, the target mode driver cannot transfer the status in write requests until the data are moved from the host system. Thus, in write requests, the IOCB messages must be exchanged one more cycle; five times in total.

## 2.2. Initiator mode operation

We use the initiator mode operation to access interior disks installed in a RAID subsystem. Each decomposed request generated by the RAID operation module is submitted to a disk queue existing on every disk, and is waiting for processing.

A disk I/O scheduler called `diskIoProc` fetches a request from its corresponding disk queue, and performs the following operations. First, it tries to obtain a mutual exclusion lock between every `diskIoProc` processes for maintaining consistency between them. Then it makes an I/O command in the form of a command IOCB, which contains SCSI CDB, a target disk number, nexus information with the firmware, etc. Finally, it puts the command IOCB on the request queue for the firmware to actually process the request.

When the firmware completes the requested command, it builds a status IOCB message on the response queue, and interrupts the system. Then the initiator mode driver responds to the interrupt, and restores nexus information from the status IOCB. If the request completes successfully, the initiator mode driver terminates the transaction by passing the status to the RAID operation module. If the request returns in failure, it will be resubmitted to the disk queue for retrying. On the other hand, if the request repeatedly fails

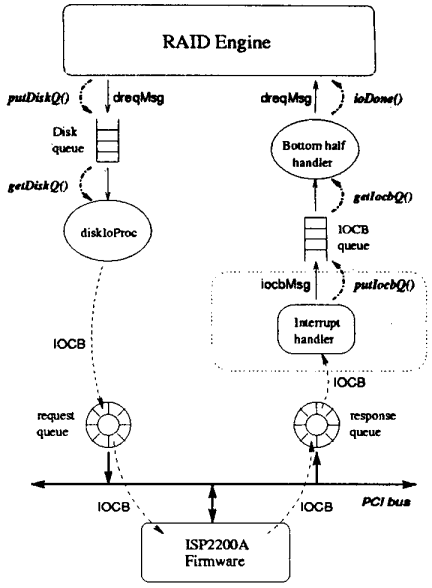



**Figure 2. Operation flow in initiator mode**

in processing, the initiator mode driver regards the corresponding disk as having been broken. The RAID operation module also switches its mode to degraded state in case of need.

The initiator operation flow which we have tried to show in this subsection is summarized in Figure 2.

# 3. Exception handling

## 3.1. Types of exception

One of the major features of a RAID subsystem is highly reliable, which is accomplished by storing error recovery information, such as a parity with data to disks. Although the RAID level-dependent module is primarily responsible for that error recovery process, it is unable to achieve successfully without a low-level I/O driver's support. Before tuning to details of the exception handling mechanism, a few remarks should be made concerning which events should be recognized to exception events and be recovered by the Fibre Channel driver.

There are two kinds of exception events which the Fibre Channel driver can be aware of and handle properly. First, a disk may be unable to respond to I/O commands issued by the driver. This situation may be caused by 1) a breakdown of the disk itself (power failure or medium error, etc), or 2) a hot removal of the disk from a FC-AL (Fibre Channel Arbitrated Loop). Secondly, the FC-AL may stall, so that all messages can not cross the loop. This may be generated by an abnormal loop circuit closing, for example, one loop

port wins arbitration, but does not return to the monitoring state.

The two situations noted above actually occurred simultaneously many times, where one disk failure may lead to the whole loop stall. More precisely, if one disk is out of order with having loop tenancy, other disks on the same loop are unable to occupy the loop, which causes the whole loop to stall. However, if the disk failure occurred without having loop tenancy, it merely affected I/O commands corresponding the disk and the whole loop stall situation did not occur.

In short, on the side of the driver, exception events to handle properly would be a disk failure or a whole loop stall situation. Detailed account of the handling mechanism for above exception events is given in the next subsection.

### 3.2. Handling mechanism

The Fibre Channel driver is not able to control the loop directly. Instead, the driver must access the loop through the firmware, exchanging IOCB messages, so that some restrictions exist in handling exception events we have already mentioned. Therefore, a recovery procedure the driver can perform is merely to call task management functions defined in the SCSI-3 architecture model, and among them we use the Target Reset task management function. However, a decision problem exists concerning the time the driver should invoke this Target Reset function. To solve this, we use a Watchdog timer mechanism supported by the VxWorks real-time operating system. By setting the Watchdog timer to each I/O command, the driver can determine which commands do not return, and recognize the occurrence of exception events.

A detailed scenario concerning how exception events are properly recovered, including the Watchdog timer mechanism, is shown below.

1. The disk I/O scheduler (`diskIoProc`) fetches I/O requests from the disk queue, and sets the Watchdog timer to the each request.

2. Then `diskIoProc` makes a command IOCB message, and issues it to the firmware.

3. At an instant, exception events such as a disk failure or a whole loop stall occur.

4. As a result, the Watchdog timeout interrupt routine is invoked after the Watchdog timer has expired. The timeout routine executes the following tasks.

   - Examine a reset flag to determine whether or not the Watchdog timer has just expired. The following processes are executed only when the reset flag is in the inactive state.
   - Reset all devices on the loop through the Target Reset task management function by issuing a Mailbox command to the firmware.
   - Activate the reset flag so as not to generate a consecutive reset.
   - Temporarily stop all `diskIoProc` processes.

5. Situation 4 leads to reset all disks on the loop, and

   - The firmware returns all executing command IOCBs with command reset status.
   - The bottom half interrupt handler gives back the I/O commands to the corresponding disk queue for retrying.
   - When all issued commands are returned from the firmware, the bottom half interrupt handler executes the following tasks.
     (a) Synchronize the driver and the firmware by issuing a Marker IOCB message to the firmware.
     (b) Inactivate the reset flag.
     (c) Restart all `diskIoProc` processes to issue I/O commands normally.

6. Return to situation 1.

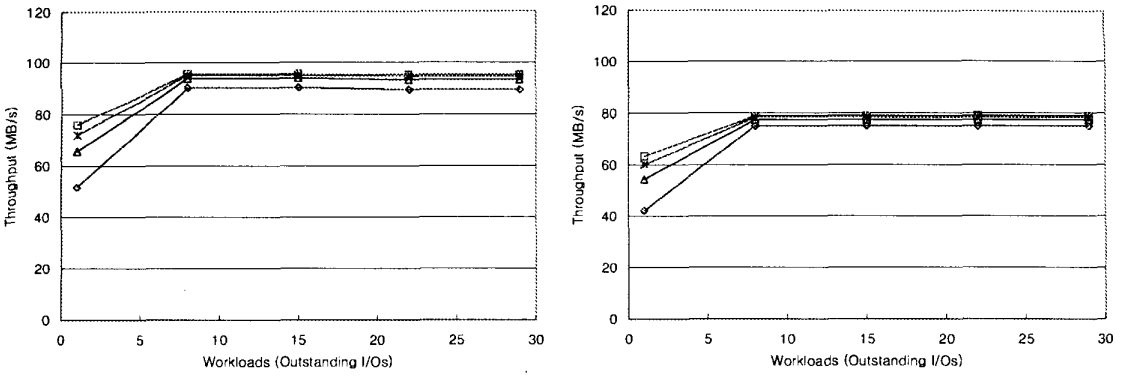
To verify the scenario works properly, some methods which generates the exception events artificially are necessary. One simple method is a hot removal of a disk from a disk enclosure which embodies the loop bypassing circuits. By performing this experiment many times, we observed that the PosRAID system successfully switched the normal state to the degraded state. Thus we conclude that our exception handling mechanism is reasonably tolerable from the exception events we have defined.

## 4. Performance evaluation

### 4.1. Experiment environment

All performance evaluation experiments were performed on an Intel Celeron processor-based PC (target system) with PosRAID software. The target system has two QLA2200 Fibre Channel adapters; one for target mode operation and the other for initiator mode operation. Each internal disk is a Seagate Cheetah 9LP model, which has 10K RPM and 1024K cache memory. In Table 1, we summarize the target system specifications.

The following measurements were performed on a Windows NT server which also has a QLA2200 Fibre Channel adapter, and were measured by the Intel IO Meter benchmark program. The workloads generated by the IO Meter




(a) Read requests

(b) Write requests

**Figure 3. Performance in target mode**

**Table 1. Target system specifications**

| | |
|---|---|
| CPU | Intel Celeron 533 MHz |
| Motherboard | AOpen AX6BC Pro |
| Memory | 128 MB |
| Internal I/O HBA | QLA2200 FC adapter (initiator mode) |
| External I/O HBA | QLA2200 FC adapter (target mode) |
| Internal disks | Four Seagate Cheetah 9LP disks, FC-AL 8-bay Enclosure |
| Network adapter | Intel EtherExpress Pro 100 |
| Operating system | VxWorks Real-time OS |

were configured to sequential access of 64 KB (◇), 128 KB (△), 192 KB (∗), and 256 KB (□) records, and the number of outstanding I/Os were 1 to 29 increased by 7 steps. All measurements show average throughput rates for the above workloads.

## 4.2. Efficiency test of the driver itself

In the following measurements, we measured the performance of the FC driver itself without a linkage of the RAID operation module. The measurements were divided into three parts; the target mode and the initiator mode performance tests, and the integration performance test of two modes.

### 4.2.1. Performance in target mode

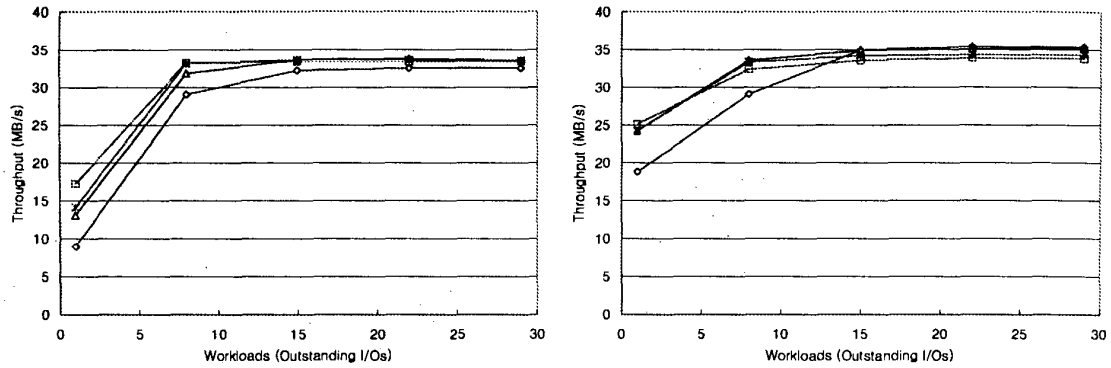
The driver performance in the target mode operation was measured by a `testTask` process which responds justly to the I/O requests received from the host system, and does not send those to the RAID operation module. In other words, the `testTask` process only allocates the memory required by the host system, and reads from or writes to that location according to the request type. There are no operations involving I/O to interior disks of the target system.

In Figure 3, the performance result is given with each workload. It shows 89.54 MB/s ~ 95.15 MB/s throughput rates in read requests. Considering the bandwidth of FC-AL is 100 MB/s, we conclude that the performance of the driver is fairly good. In write requests, we observe that average throughputs are 75.00 MB/s ~ 78.78 MB/s, which is less than in the case of read requests. The reason is that the driver must exchange IOCBs one cycle more with the firmware in write requests.

### 4.2.2. Performance in initiator mode

The driver performance in the initiator mode operation was measured by the test shell command mode which is a part of the RAID management module. For following measurements, we added a test shell command called `perf`, which performs I/O to its interior disks in various request forms and shows a response time and an average throughput rate.

First, we measured the average throughput rates according to the increase in the number of interior disks. Table 2 shows these measurement results for a sequential access of 64 KB I/O requests. From the results, we can know the fact that the maximum throughput of one interior disk is about 18 MB/s. This is also verified through QLogic's official Windows NT driver. Furthermore, we see that the throughput steadily increases in direct proportion to the increase in the number of disks. Therefore we recognize that we must




(a) Read requests

(b) Write requests

**Figure 4. Integration performance of two modes (1): 64 KB stripe unit size, 4 disks**

use the proper number of disks to achieve a good performance.

**Table 2. Throughput vs. number of disks (MB/s)**

| I/O requests | Number of disks | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Read requests | 18.244 | 36.488 | 54.669 | 72.617 |
| Write requests | 18.188 | 36.364 | 54.483 | 72.727 |

Next, we measured the Watchdog timer overhead by performing in the same testing environment while removing the Watchdog timer function. Comparing Table 3 and Table 2, the latter is slightly better, but the differences is slight. Thus we can see that the Watchdog timer overhead is insignificant. This indicates the suitability of our solution to adopt a Watchdog timer mechanism to recover exception events.

**Table 3. Throughput without the Watchdog timer (MB/s)**

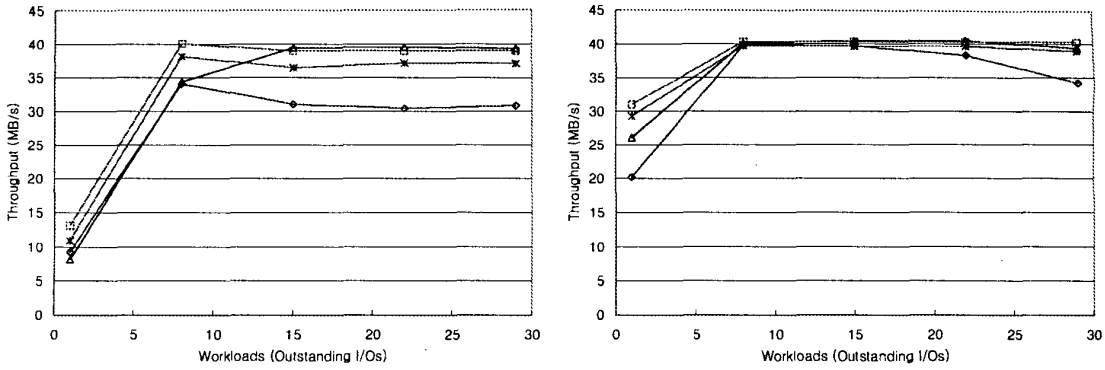
| I/O requests | Number of disks | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Read requests | 18.251 | 36.502 | 54.711 | 72.837 |
| Write requests | 18.174 | 36.364 | 54.524 | 72.672 |

#### 4.2.3. Integration performance of two modes

We measured the integration performance of two modes by driving the target mode and the initiator mode operations simultaneously. For this examination, we changed the function of the `testTask` process to issuing I/O requests received from the host system to the interior disk I/O module directly. That is, we make the `testTask` process execute a striping (RAID level 0) operation itself without the support of the RAID operation module.

To begin with, we examined the result for the internal stripe unit size by 64 KB using 4 interior disks. In Figure 4, the performance for the workloads mentioned previously is given. It shows 32.58 MB/s $\sim$ 33.49 MB/s throughput rates in read requests, and 33.86 MB/s $\sim$ 35.34 MB/s in write requests. Next, we measured the performance for the internal stripe unit size by 128 KB using 4 interior disks, shown in Figure 5. Its throughput rates are 30.79 MB/s $\sim$ 39.29 MB/s in read requests, and 34.17 MB/s $\sim$ 40.24 MB/s in write requests. As the results indicate, the integration performance of the two modes is much lower than that of each mode; about 50% degradation of the performance of the initiator mode operation.

To examine a cause about the performance degradation, we execute the following testing. The next measurement was conducted in the same environment as above, but we measured the performance each mode separately without the integration. Table 4 shows these measurement results. From the table, it appears that the performance degradation phenomenon recurs even if we drive the two mode operations separately. In addition, the total throughputs of each workload are almost equal to 77 MB/s. This results indicate that the performance degradation is caused not by the integration of the two modes but by other issues we must




(a) Read requests

(b) Write requests

**Figure 5. Integration performance of two modes (2): 128 KB stripe unit size, 4 disks**

investigate.

**Table 4. Performance degradation analysis (MB/s)**

| Measurement patterns | Workloads | | | |
|---|---|---|---|---|
| | 1 | 4 | 8 | 16 |
| Target mode (1) | 24.53 | 33.39 | 32.87 | 33.66 |
| Initiator mode (2) | 52.02 | 44.13 | 44.14 | 44.07 |
| Total throughput (1+2) | 76.55 | 77.52 | 77.01 | 77.73 |

One possible explanation is that a PCI bus bottleneck leads to the performance degradation. Figure 6 shows the interior architecture of the target system briefly. The diagram shows that two Fibre Channel HBA and one LAN adapter share one PCI bus. Therefore, it is possible that heavy overheads caused by PCI bus arbitration may happen when each adapter simultaneously operates as a PCI master. Although the bandwidth of a PCI bus is theoretically 133 MB/s, we observe that the PCI bus arbitration overhead is very high, since total throughputs remain at approximately 77 MB/s. In conclusion, the only method to solve this performance degradation problem is changing the target system PCI architecture fundamentally; one PCI bus for each Fibre Channel HBA.

## 5. Conclusion

We have designed and implemented a Fibre Channel network driver for SAN-attached RAID controllers in a VxWorks real-time operating system. We have described the target and initiator mode operations in turn. To support the fault resilient operation of the RAID system, we have defined exception events to be recovered, and designed the handler of those. From the performance measurement results, we determined that the maximum I/O throughput reaches 95 MB/s in the target mode operation and 79 MB/s in the initiator mode operation. As we used the commodity PC motherboard as a RAID controller, we conclude that the Fibre Channel driver achieved a reasonably good performance.

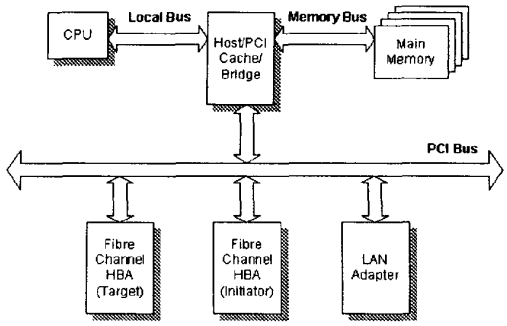



**Figure 6. PCI bus architecture**

## References

[1] D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD*, June 1988.

[2] P. Massiglia, "Fibre Channel, Storage Area Networks, and Disk Array Systems," *White Paper*, Adaptec Inc., April

1998.
[3] K. Amiri, G. Gibson, and R. Golding, "Highly Concurrent Shared Storage," *Proceedings of International Conference on Distributed Computing Systems*, April 2000.
[4] M. Sachs, *et. al.*, "LAN and I/O Convergence: A Survey of the Issues," *IEEE Computer*, December 1994.
[5] G. Gibson, *et. al.*, "A Cost-effective, High-bandwidth Storage Architecture," *Proceddings of the 8th Conference on ASPLOS*, 1998.
[6] R. Bhoedjang, *et. al.*, "User-level Network Interface Protocols," *IEEE Computer*, November 1998.
[7] P. Cheng, *et. al.*, "Design of High Performance RAID in Real-time System," *ACM Computer Architecture News*, Vol. 27, No. 3, June 1999.
[8] QLogic Corp., *ISP2200/2200A Firmware Interface Specification.* October 1999.
[9] Z. Meggyesi, "Fibre Channel Overview," http://www.cern.ch/hsi/fcs/spec/overview.htm
[10] *Fibre Channel – Physical and Signaling Interface (FC-PH)*, ANSI X3.230, Rev 4.3, 1994.
[11] *Fibre Channel Protocol for SCSI (FCP)*, ANSI X3.269, Rev 12, 1995.
[12] *Fibre Channel – Arbitrated Loop (FC-AL)*, ANSI X3.272, Rev 4.5, 1995.
[13] T. Clark, *Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel SANs*. Addison Wesley Longman, 1999.
[14] G. Stephens, J. Dedek, *Fibre Channel: The Basics*. ANCOT Corp., 1997.
[15] R. Kembel, *Arbitrated Loop*. Northwest Learning Associates, 1996.
[16] "Development of a RAID System Software for Mid-Range Servers," *Technical Report, CSE-SSL-2000-005-A*, System Software Laboratory, POSTECH, August 2000.