내장형 리눅스 기반 RAID 시스템의 구현 및 성능평가

백 승 훈† · 박 종 원††

요 약

본 논문에서는 내장형 리눅스를 이용한 RAID 시스템의 하드웨어 및 소프트웨어 구현 방법을 제시하고 성능을 측정하여 본 시스템의 장단점을 제시한다. 파이버 채널 디스크와 호스트와의 정합을 위하여 세 개의 파이버 채널 제어기를 포함하는 하드웨어를 설계 제작하였으며, 이 하드웨어 위에 내장형 리눅스를 이식하고 RAID 소프트웨어를 구현하였다. RAID 기능을 위하여 SCSI 목표 모드 디바이스 드라이버와 목표 모드 SCSI 모듈을 구현하여 호스트 컴퓨터에게 SCSI 블록 디바이스를 제공한다. 또한 RAID의 기능을 위하여 리눅스의 Multi-device 모듈을 사용하였고, 높은 성능을 제공하기 위하여 Multi-device 모듈과 목표 모드 SCSI 모듈의 사이에서 연동하는 데이터 캐쉬 모듈을 구현하였다. 리눅스의 RAID 5 모듈을 수정하여, 읽기 성능을 대폭 향상시켰다. 벤치마크는 새로운 RAID 5모듈이 기존의 방법보다 전체적인 성능에서 우수함을 보여준다.

An Implementation and Performance Evaluation of a RAID System Based on Embedded Linux

Sung-Hoon Baek† · Chong-Won Park††

ABSTRACT

In this article, we present, design, and implement a software and hardware for an embedded RAID system. The merits and drawbacks of our system are presented by performance evaluation. The proposed hardware system consists of three fibre channel controllers for the interface with fibre channel disks and hosts. Embedded Linux in which a RAID software is implemented is ported to the hardware. A SCSI target mode device driver and a target mode SCSI module are designed for that our RAID system is considered as a block device to a host computer. Linux Multi-device is used as RAID functions of this system. A data cache module is implemented for high performance and the interconnection between Linux Multi-device and the target mode SCSI module. The RAID 5 module of Multi-device is modified for improvement of read performance. The benchmark shows that the new RAID 5 module is superior to the original one in overall performance.

키워드 : 독립디스크중복배열(Redundant Arrays of Inexpensive Disks ; RAID), 내장형 리눅스(Embedded Linux)

1. 서 론

최근 컴퓨터의 계산처리 능력의 향상과 개인 디스크의 대용량화로 인하여 개인이 대용량의 고화질 고음질을 지원하는 멀티미디어 파일을 개인 컴퓨터에 소유하게 되었다. 이러한 소비자의 욕구를 충족시키기 위하여 인터넷 서비스 제공 업체는 초대용량의 멀티미디어 컨텐츠들을 보유할 수 있는 대용량 데이터 저장 장치들을 필요로 하게되었다.

인터넷 서비스 제공업체는 폭발적으로 증가하는 사용자 접속을 지원하기 위하여 컴퓨팅 노드의 클러스터링을 사용하였다. 대용량의 멀티미디어 데이터를 공급하기 위하여 클러스터 노드들은 SAN(Storage Area Network)을 통하여 대용량 초고속의 RAID를 사용한다. SAN이란 연결된 서버에 관계없이 원거리에 분산된 저장 장치들을 파이버 채널(Fibre Channel)[1] 네트워크로 연결되어 SAN에 연결된 네트워크 노드들이 저장 장치들을 공유할 수 있도록 해주는 초고속 통신망이다. SAN은 고성능(high performance), 고확장성(high scalability), 공유성(sharability) 및 고가용성 (high reliability)을 제공한다. 파이버 채널은 2Gbps의 직렬 전송방식을 갖기 때문에 루프구조에서는 126개의 디스크를 연결할 수 있다. 또한 연결 케이블 길이가 10km까지 확장 가능하므로, 디스크 개수와 케이블 길이 제약을 갖는 SCSI를 대체하고 있다[2].

독립디스크중복배열(RAID : Redundant Arrays of Independent Disks)은 고가의 대형 디스크를 사용하지 않고 일반

† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 컴퓨터시스템연구부 연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 컴퓨터시스템연구부 책임연구원
논문접수 : 2002년 9월 30일, 심사완료 : 2002년 11월 28일

적인 디스크들을 다중으로 사용하여 용량과 성능을 향상시킨다. RAID에서 디스크들을 다량으로 사용함으로써 발생되는 시스템의 신뢰성 문제를 해결하기 위하여 디스크 미러링, 패러티 디스크를 사용한다[3].

하지만, SAN과 RAID는 설치비용을 많이 필요로 하기 때문에, 소프트웨어로써 스트라이핑, 미러링, 패러티 디스크를 구현하여 비용을 절감하는 방법도 사용된다. Solaris는 "Solaris Management Console"의 "Enhanced Storage" 도구를 이용한 볼륨 관리를 통하여 RAID 0(stripe), RAID 1(mirror), RAID 5(striped parity)를 설정할 수 있다. 리눅스에서는 커널에 Multi-device 모듈을 포함시키면, 논리 볼륨 관리자 및 RAID0-1-4-5를 별도의 하드웨어 없이 여러 개의 디스크만 있으면 구성할 수 있다.

본 논문에서는 내장형 리눅스 시스템을 이용하여 RAID 시스템의 설계 및 구현을 제시한다. 이 시스템은 ARM 5TE 명령과 호환되는 Intel의 XScale 80200(733MHz) 프로세서를 사용하고, 운영체제로서 내장형 리눅스 시스템을 채택했으며, RAID 기본 기능을 위하여 리눅스에 포함된 Multi-device 모듈[4, 5]을 사용하였고, 디스크 및 호스트 인터페이스로써 파이버 채널(Fibre Channel)을 사용하였다. 본 시스템의 구현을 위해서 추가적으로 호스트로부터의 요구를 수용하는 파이버 채널 목표(target) 디바이스 드라이버와 호스트의 SCSI 명령을 분석하는 목표 SCSI 모듈과 데이터 캐쉬 제어 모듈을 구현하였다.

2. 내장형 리눅스

내장형 리눅스(Embedded Linux)란 특수 목적에 사용되는 마이크로 프로세서를 가지는 시스템을 구동하기 위하여 범용 컴퓨터에는 불필요한 요소를 제거하고 내장형 시스템에 필요한 요소를 추가한 운영체제이다. 내장형 시스템과 범용 컴퓨터의 가장 쉬운 차이점은 사용자 인터페이스이다. 범용 컴퓨터는 모니터, 키보드, 마우스 및 하드디스크 등과 같은 장치를 가지지만, 내장형 시스템은 이런 것들을 전혀 가지지 않고 간단한 버튼이나 터치 스크린 및 직렬 통신 장치를 가지며, 데이터를 수집 및 전송하고 명령을 받아들이고, 네트워크를 모니터링하는 일을 수행한다. 내장형 시스템에 키보드와 모니터를 설치할 수 있으나 일반 동작 모드에서는 불필요한 것이 될 수 있다. 하지만 이것들은 디버깅 용도로 사용될 수도 있다.

리눅스의 가장 큰 장점이 공개된 소스라는 것은 논란의 여지가 없다. 어떠한 사용료를 지불할 필요 없이 리눅스를 기반으로 제품을 개발하고 또는 제품 개발을 위하여 리눅스의 일부를 수정할 수 있다. 단 한가지 제한은 개발을 위해서 수정한 부분을 세상에 공개해야 하는 것이다. 반면 WindRiver나 QNX에서 제공하는 상용 운영체제를 사용하게 된다면 개발을 시작하기 위해 수만 달러의 비용이 필요하게 된다. 또한 매 제품이 팔릴 때마다 사용료를 지불해야 한다.

소스가 공개되어 있다는것은 개발에 상당한 편의를 제공한다. 만약 디버깅 도중에 공개되지 않은 코드로 접근하면 개발자는 큰 벽에 부딪히게 된다. 버그가 공개되지 않은 소스에 있을 수도 있고, 대부분은 사용자의 소스에 있을 수도 있다. 만약 사용자 소스에 버그가 있는 경우에 디버깅을 위하여 많은 시간을 소비할 수 있지만, 소스가 공개되어 있다면 사용자 소스의 문제를 쉽게 발견할 수 있다. 그 반대의 경우라면 쉽게 버그를 발견하고 사용자가 소스를 직접 고칠 수도 있고 그 코드의 개발자에게 고치도록 도움을 요청할 수도 있다. 리눅스 드라이버를 개발하는 경우 특별한 기능을 구현하는 것도 쉽다. 공개된 리눅스 드라이버의 소스가 어떻게 그 기능을 구현하였는지 보고, 그 소스를 복사한 뒤 필요한 부분만 수정하면 된다. 마지막으로 대게 공개 소프트웨어는 문서화가 잘 되어 있지 않지만, 리눅스 문서 프로젝트를 이용하여 많은 무료 리눅스 문서를 얻을 수 있다[6].

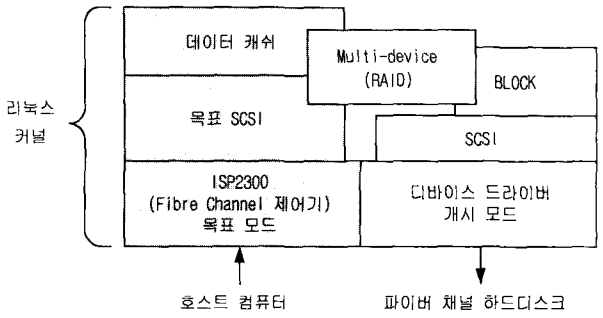
하지만 소스 공개의 단점도 있다. 리눅스의 공개 소스 정책에 따라서 리눅스 공개 소스를 사용하고, 이것을 수정하였을 경우에는 사용료 없이 수정된 소스도 공개하여야 한다. 그러나 경쟁업체를 앞서나가기 위해서 소스를 공개하려하지 않는 많은 경우가 있다. 소스를 공개하려하지 않는다면 프로젝트에 어떠한 공개 소스도 포함하지 않아야 한다. 커널 버전이 자주 바뀌고 발표되는 것도 사용자에게 혼돈을 가져다 준다. 또한 악의를 가진자가 공개된 소스를 분석하여 시스템의 취약한 부분을 시스템 해킹 목적으로 사용하는 경우도 있다. 리눅스 시스템의 버퍼 오버플로우 공격이 대표적인 경우이다[7].

3. RAID 시스템의 구현

RAID 시스템은 내장형 하드웨어를 갖는, RAID 기능을 갖춘, 시스템이다. 즉, 호스트 컴퓨터와 SCSI나 파이버 채널과 같은 정합장치로써 연결되어 호스트 컴퓨터에게 SCSI 장치로 제공된다. 또한 다중의 디스크들과 연결하기 위한 SCSI나 파이버 채널과 같은 정합장치를 추가로 필요로 한다. RAID 시스템이 높은 처리량과 높은 대역폭을 갖추기 위해서는 호스트 컴퓨터에서 보내어지는 명령을 순차적이지 않고 병렬적으로 처리해야 한다.

내장형 RAID 시스템을 구현하기 위해서, 높은 성능의 처리 능력을 가지는 마이크로 프로세서와 디스크를 제어할 하드웨어 시스템을 구축하고, 운영체제를 로딩할 부트로더 및 내장형 리눅스의 이식과 RAID 기능소프트웨어가 필요

하다. 이를 위하여 호스트 컴퓨터와의 정합을 위한 파이버 채널 목표 모드 디바이스 드라이버와, 호스트 컴퓨터로부터 내려오는 SCSI 명령을 처리할 목표 모드 SCSI 모듈과, 성능 향상을 위한 대용량 데이터 캐쉬 모듈을 구현하였다. 각 RAID 레벨 기능은 리눅스의 Multi-device 모듈을 대부분 이용하고, 성능 향상을 위하여 읽기 캐쉬를 수정함으로써 읽기 성능을 대폭 개선하였다.



(그림 1) 내장형 리눅스를 이용한 RAID 시스템의 소프트웨어 구조

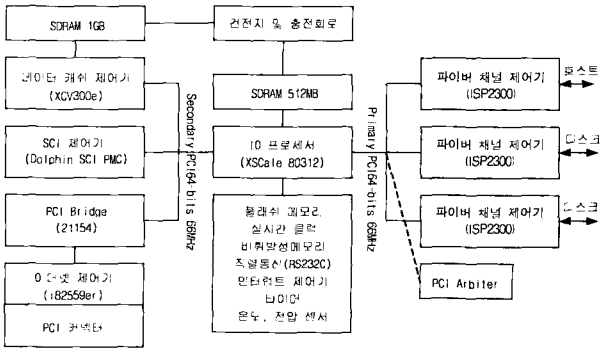
(그림 1)은 내장형 리눅스를 이용한 RAID 시스템의 소프트웨어 구조를 보여준다. 이 그림에서 회색 음영으로 표시된 모듈은 새로 구현한 부분이고, 흰색의 모듈은 리눅스에서 제공되는 것이다. 파이버 채널로 연결된 호스트 컴퓨터에서 보내지는 SCSI 명령은 ISP2300 디바이스 드라이버의 목표 모드 모듈에 의해서 하드웨어 제어기에서 목표 모드 SCSI 모듈로 전송된다. 목표 모드 SCSI 모듈은 데이터 캐쉬의 내용을 읽기/쓰기를 하거나 Multi-device로 읽기/쓰기를 한다. 데이터 캐쉬 모듈은 최근에 사용된 데이터 블록을 목표 모드 SCSI에 제공한다. 목표 모드 SCSI 모듈은 쓰기 요청 받은 것을 데이터 캐쉬로 전달하고, 디스테이지(Destage) 정책[8]에 따라 더티 블록을 Multi-device로 쓰기 명령을 내린다.

디스테이지 정책이란, 쓰기 요청에 의한 데이터는 캐쉬로 보내어지고, 그 이후의 어느 시점에서 디스크로 더티 블록을 전송하는데, 높은 성능을 위하여 이 쓰기 시점을 정하는 정책을 말한다.

Multi-device 모듈은 Linear, RAID 0, RAID 1, RAID 5의 기능을 수행하는 RAID 기능과 논리 볼륨 관리자의 기능을 지원한다. 호스트로 보내어지는 데이터는 최종적으로 Multi-device, block, SCSI, ISP2300 개시(Initiator) 모드 디바이스 드라이버를 각각 거쳐서 최종적으로 하드디스크로 전송된다.

3.1 시스템의 구성

본 논문에서 제시하는 내장형 리눅스를 이용한 RAID 시스템의 하드웨어 구성도를 살펴보면 (그림 2)와 같다.



(그림 2) 내장형 RAID의 하드웨어 구성도

RAID는 고속의 입출력을 처리하기 위해서는 반드시 고성능의 중앙처리장치를 요구하기 때문에, 본 시스템은 중앙처리장치로서 인텔 80200 XScale[9] 733MHz 마이크로 프로세서와 두 개의 64 비트 66MHz PCI 버스와 100MHz SDRAM를 지원하는 80310 입출력 칩셋으로 구성된 IOP 80312[9]를 채택하였다. 그리고 200MB/s 대역폭의 파이버 채널 디스크의 대역폭과 호스트 컴퓨터로 데이터를 전송하는 파이버 채널의 200MB/s 대역폭을 동시에 수용하기 위하여 528MB/s 대역폭의 64 비트 66MHz PCI를 선택하였다.

주 PCI 버스에는 총 세 개의 파이버 채널 제어기가 있는데 하나는 호스트와의 정합 - 호스트 컴퓨터에게 본 시스템이 블록 디바이스로 보이게 하는 정합 - 으로 쓰이고, 나머지 두 개는 디스크와의 정합으로 사용된다. 중복적인 디스크 정합은 높은 대역폭과 가용성을 높인다. 보조 PCI 버스에는 대용량의 데이터 캐쉬를 지원하기 위한 데이터 캐쉬 제어기와 RAID 제어기간의 통신을 위한 SCI(Scalable Coherent Interface) [10] 제어기와 100Mbps 이더넷 제어기가 있다. 데이터 캐쉬 제어기는 1G바이트의 데이터를 저장할 수 있고, 패리티 블록 계산을 위한 배타적 논리합 연산기능을 가지고 있다. 이더넷은 RAID의 환경 설정과 관리를 인터넷을 통하여 수행하기 위하여 존재하지만 개발과정에서 프로그램의 실행코드를 고속으로 수신할 수 있는 좋은 디버깅 포트로 이용되어 편리하다. 개발과정에서 SCSI 호스트 버스 어댑터를 여분의 PCI 커넥터에 설치할 수 있어서, 지역 하드디스크에 루트 파일시스템을 둘 수 있었다. 그럼으로써 커널 로딩 시간을 대폭 줄였다. 그런데 이 PCI 커넥터는 개발의 편의를 위한 것이며 목표 시스템의 용도는 없다.

전원이 꺼지는 경우에도 캐쉬 메모리에 있는 데이터가 휘발되는 것을 막기 위하여 건전지 및 충전회로로써 지역 메모리와 캐쉬 메모리에 전원을 공급한다. 실시간 클록은 RAID관리나 오류기록의 시간을 정확히 알리기 위해서 사용되고, 직렬통신은 리눅스 콘솔로 사용되고, 프로세서 스케줄링을 위한 타이머와, 하드웨어 진단을 위한 온도, 전압, 및 팬 속도 센서를 가지고 있다.

3.2 부트로더 및 내장형 리눅스 이식

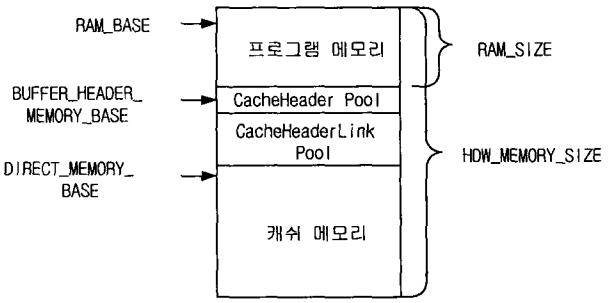
내장형 리눅스 이식에 앞서 부트로더를 이식하여야 한다. 내장형 리눅스는 하드웨어를 초기화하는 기능이 없기 때문에 부트로더를 필요로 한다. 부트로더는 영구적으로 플래시 메모리나 롬영역에 저장되어 있고, 시스템에 전원이 켜질 때에 맨 처음 수행되는 부분이다. 부트로더는 맨 처음 지역 메모리를 감지 및 초기화하고 주요 하드웨어를 초기화한다. 그 뒤에 플래시 메모리나 직렬 통신장치나 이더넷을 통하여 내장형 리눅스의 커널과 파일시스템 이미지를 지역메모리로 가져와 실행시킨다. 또한 부트로더는 여러 종류의 통신장치를 통하여 리눅스 이미지를 플래시 메모리에 저장할 수 있는 기능도 가지고 있다. 이러한 기능은 제품이 판매된 후에 버전 업그레이드를 위한 기능으로도 사용된다.

최종 목표시스템은 통신장치를 통하여 운영체제를 로딩하지 않고 이전에 저장한 플래시 메모리에서 직접 램으로 로딩한다. 최종 목표 시스템에는 플래시 메모리에 리눅스이미지가 있어야 하지만, 시스템 커널을 수정하는 개발을 한다면 커널을 이더넷을 통하여 수신하는 것이 개발 시간을 단축시킨다. 부트로더는 하드웨어의 개발과정에서 하드웨어 디버깅이나 간단한 프로그램 시험용으로 사용되는 모니터 프로그램의 용도로도 이용된다. 본 시스템의 부트로더로서 GNU의 RedBoot[11]를 사용했다.

내장형 리눅스를 사용할 때에는 개발 목표 시스템의 보드 지원 패키지(BSP)를 지원하는 개발 지원 패키지를 구입하는 것이 개발 시간을 단축시킨다. 본 시스템에서는 BlueCat 리눅스 4.0[12] XScale 버전을 사용하였다. 이것은 RAID의 필수인 디스크 핫 플러그를 지원하는 리눅스 커널 2.4[13]를 사용하고 있으며, 리눅스 소스와 보드 지원 패키지 및 컴파일러와 같은 개발도구와 리눅스 이식에 관한 개발 지원을 온라인으로 받을 수 있다. XScale은 ARM과 호환이 되기 때문에 ARM용 GNU 개발 도구를 사용하였다.

3.3 내장형 리눅스를 이용한 RAID 소프트웨어 구조

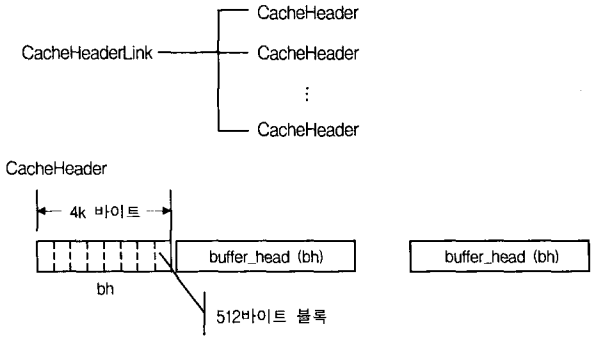
RAID는 호스트 컴퓨터에서 송수신된 데이터를 캐쉬하는 대용량의 캐쉬 메모리를 가져야 한다. 그러나 리눅스 커널에서는 물리 메모리의 절반이상을 할당하는 내부 함수가 없으므로 커널을 구성하는 초기에 대용량의 캐쉬 메모리를 할당해 두어야 한다. (그림 3)과 같이 캐쉬 메모리와 이를 제어하는 캐쉬 헤더들이 고정된 주소에서 사용된다. 실제 크기가 HDW_MEMORY_SIZE인 물리 메모리를 커널에게 RAM_SIZE 크기만으로 인식시켜서, 그 나머지 영역을 커널이 사용하지 않게 하여 이 공간을 캐쉬 메모리와 제어를 위한 공간으로 사용할 수 있다. 그러나 이 나머지 공간은 가상 메모리를 위한 페이지 테이블이 존재하지 않으므로, (그림 3)의 아래쪽에 나타난 구조체로써 페이지 테이블을 할당한다. 이 구조체는 리눅스 소스의 mm.c 파일에 정의되어 있는 것으로서 커널을 초기화할 때에 캐쉬의 가상 메모리 페이지 테이블을 할당한다.




```
static struct map_desc iq80310_io_desc [] __initdata =
{
   ...
 { BUFFER_HEADER_MEMORY_BASE, BUFFER_HEADER_MEMORY_START,
BUFFER_HEADER_MEMORY_SIZE, DOMAIN_KERNEL, 1, 1, 1, 0},
 { DIRECT_MEMORY_BASE, DIRECT_MEMORY_START,
DIRECT_MEMORY_SIZE, DOMAIN_KERNEL, 1, 1, 1, 0},
   LAST_DESC
};
```

(그림 3) 메모리 구조

(그림 4)는 캐쉬를 관리하기 위한 CacheHeaderLink와 CacheHeader 자료형을 보여준다. CacheHeaderLink는 여러 개의 CacheHeader와 bh를 보유하고 하나의 호스트 요구에 필요한 블록을 관리하기 위한 단위이며, 호스트 요구 단위로 생성 및 소멸된다. CacheHeader는 호스트 요구와 관계없이 존재하며 캐쉬를 관리하기 위한 기본 단위 자료형이다. buffer_head는 실제 디스크 또는 Multi-device 모듈로 명령을 내려보낼 때에 필요한 자료형이다. 디스크로 쓰기/읽기 명령을 내보내기 위해서는 리눅스 소스의 ll_rw_blk.c에 정의된 generic_make_request() 함수를 이용한다. 이 함수는 buffer_header 형의 구조체를 인자로 취한다. 이 인자




```
typedef struct CacheHeaderLink_tag {
    struct CacheHeader_tag *pCH ;
    struct CacheHeaderLink_tag *next ;
    struct buffer_head bh[CACHE_NUM_BUFFER_HEADER] ;
} CacheHeaderLink ;
```

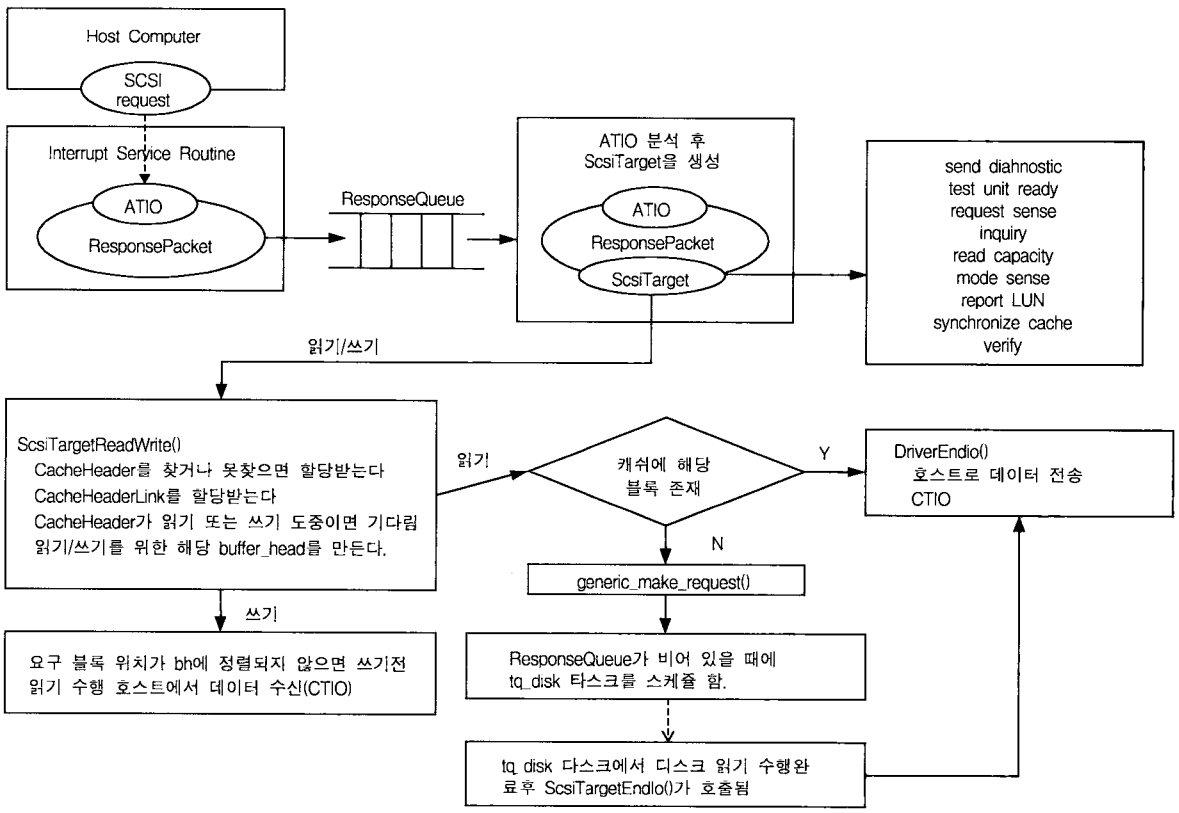
(그림 4) 캐쉬 헤더 구조

는 장치 번호, 섹터 위치, 읽을 섹터의 양 등의 속성을 포함한다.

(그림 5)는 호스트 컴퓨터에서 내려오는 서비스 요청에 따른 수행 흐름도를 보여준다. 호스트 컴퓨터에서 SCSI 요구가 파이버 채널을 타고 RAID 시스템으로 전송되면, 파이버 채널 제어기는 호스트의 SCSI 명령을 ATIO 자료형으로 만든 뒤에 인터럽트를 발생시킨다. 인터럽트 서비스 루틴에서는 제어기에서 ATIO 자료를 추출하고 이것을 새로 할당한 ResponsePacket 패킷에 복사하고 ResponseQueue 큐에 넣고, 커널 쓰레드가 이 큐에서 패킷을 읽어 가도록 커널 쓰레드를 깨운다. 이 쓰레드는 RAID 운영을 위하여 디바이스 드라이버에서 새로 생성한 쓰레드이다. 그러면 그 쓰레드는 큐에서 ResponsePacket 패킷을 빼내고 SCSI 명령이 담겨있는 ATIO 자료를 분석하여 ScsiTarget 자료형에 맞게 변환하여 각각의 기능별로 목표 모드 SCSI 모듈로 함수들을 호출한다. 목표 모드 SCSI 명령은 읽기/쓰기 외에 "send diagnostic," "test unit ready," "request sense," "inquiry," "read capacity," "mode sense," "report LUN," "synchronize cache," "verify" 명령들을 기본으로 처리해야한다.

읽기나 쓰기 명령의 경우에는 해당 블록들을 캐쉬에서 먼저 찾아 캐쉬 데이터를 담고 있는 CacheHeader를 얻는다. 그 CacheHeader가 읽기 또는 쓰기 도중이면 마칠 때까지 기다리고 그럴 수 없는 경우엔, 빈 CacheHeader를 할당받는다. 빈 CacheHeader가 없다면 가장 오래된 CacheHeader를 비워서 할당한다. 비울 CacheHeader에 더티 블록이 있으면 해당 블록을 디스크로 쓴 뒤에 비워서 할당한다. 그리고 CacheHeader의 개수만큼 CacheHeaderLink를 할당한다. CacheHeaderLink는 하나의 요구에 필요한 여러 CacheHeader들을 연결하는 역할을 하고, 디스크로 명령을 보내기 위해서 필요한 buffer_header를 가진다. CacheHeaderLink는 요구 처리 완료 후에 해제되지만 CacheHeader는 캐쉬 영역에서 해제되지 않는다.

읽기의 경우에는 캐쉬에 해당 블록이 존재하면 파이버 채널 디바이스 드라이버의 DiverEndIo()를 호출한다. 이 함수에서는 제어기로 CTIO 명령을 보내어 호스트로 데이터를 전송하고 "tq_disk" 타스크를 스케쥴링하거나 새로운 요구가 있을 때까지 자게 된다. 캐쉬에 해당 블록이 없으면 generic_make_request() 함수로 읽기 명령을 보낸다. 이 때에 이 함수의 목표 디바이스는 RAID 0, RAID 5등의 기능을 수행하는 Multi-device 모듈이 된다. 그 다음 ResponseQueue 큐가 비어 있지 않을 때까지 "tq_disk" 타스크를 스케쥴링한다. 실제의 디스크의 읽기/쓰기는 "tq_disk" 타스크에서 일어나고 완료 후 비동기로 콜백 함수가 호출된다. 본 시스템에서는 ScsiTargetEndIo() 함수가 콜백되어 호출된다. 그러면 디바이스로부터 읽은 데이터는 CacheHeader에 의해서 관리되어 캐쉬되고, 디바이스 드라이버로 CTIO에



(그림 5) 서비스 요청에 따른 수행 흐름도

의해서 호스트로 전송된다.

쓰기의 경우는 요구 블록의 위치가 buffer_head에 정렬되지 않으면, 즉 4k바이트에 정렬되지 않으면 쓰기 전 읽기를 수행한다. 왜냐하면 4k바이트로 디스크로 읽기/쓰기를 하지만 호스트로의 요구는 1024바이트로 수행하기 때문이다. 예를 들어서 2번 블록에 한 블록(1024바이트)의 쓰기 요구가 들어오면, buffer_head에는 0,1,3번 블록에는 무효한 데이터 값을 가진다. 하지만 buffer_head 단위로 디스크에 쓰기가 일어나므로, 일부 무효한 블록을 가지는 buffer_head로는 디스크로 쓰기를 할 수 없다. 즉 먼저 buffer_head에 유효한 데이터를 채우기 위해서 디스크로부터 4k바이트를 읽은 뒤에 호스트 데이터 1024바이트를 buffer_head에 쓰고, 디스크로 총 4k바이트로 쓰기 수행을 한다. 하지만 실제로 디스크로 쓰는 동작은 캐쉬 디스테이지 알고리즘[8]에 의해서 나중에 수행된다.

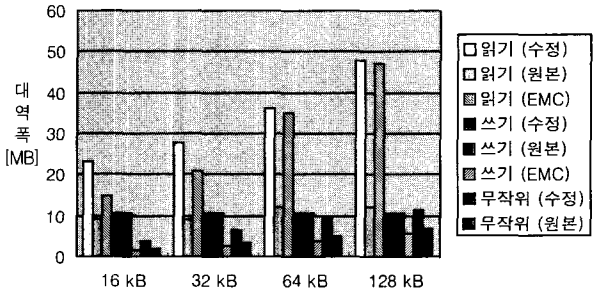
캐쉬에 있는 더티 블록들을 디스크에 쓰는 동작은 복수개의 CacheHeader단위로 일어난다. 복수개의 CacheHeader 단위의 쓰기는 대량의 데이터를 한꺼번에 쓰기함으로써 RAID 5에서 배타적 논리합 연산을 최소화하는 효과가 있다. 한꺼번에 쓰는 블록 크기와 하나의 stripe를 이루는 크기(strip 크기와 데이터 디스크 개수의 곱)가 같을 때에 최적의 성능이 나온다. 호스트로부터 아무런 동작이 없는 상태로 10초가 유지되면 모든 더티 블록들을 디스크로 쓰기 시작하고, 쓰기 도중에 호스트로부터의 입력이 있으면 중지된다. 또한 캐쉬 디스테이지 알고리즘으로서 하이/로우 마크 알고리즘이 사용된다. 이 알고리즘은 더티 블록의 캐쉬 이용률이 70%이상이 되면 30%이하가 될 때까지 더티 블록의 디스크 쓰기를 수행한다[8]. 디스크 쓰기는 이 외에 다른 경우에도 발생한다. 빈 CacheHeader의 요구가 있는데 빈 CacheHeader가 없고 가장 오랫동안 사용하지 않은 CacheHeader가 더티이면, 이것의 쓰기를 수행한다.

3.4 리눅스 RAID 5 모듈의 성능 개선

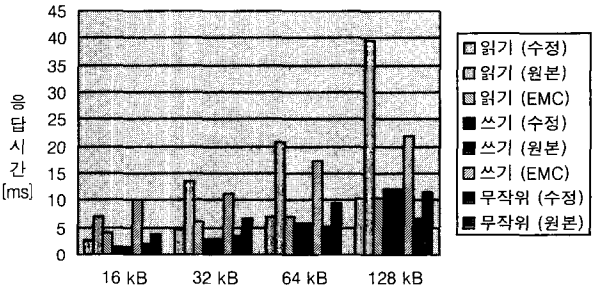
본 시스템에서는 리눅스의 RAID 5 모듈의 읽기 성능을 개선하였다. 리눅스의 Multi-device 모듈은 작은 쓰기의 배타적 논리합 연산에 필요한 블록을 읽는 것을 줄이기 위하여, 시스템 블록 캐쉬 외에 RAID 5 모듈은 모든 읽은 데이터를 자체적으로 캐쉬한다. 그러나 작은 쓰기를 위한 읽기 캐쉬의 히트율은 매우 낮다. 또한 읽기 캐쉬에 의해서 유발되는 총 두 번의 메모리 접근은 시스템의 성능을 저하시킨다. 왜냐하면 자체 캐쉬를 위한 메모리 복사가 읽기와 쓰기 메모리 접근을 발생시키기 때문이다. 본 시스템에서는 이러한 RAID 5 모듈 내의 읽기 캐쉬를 제거하여 성능을 향상시켰다. 또한 수정된 RAID 5 모듈은 임의의 입출력 부하에도 더 높은 성능을 보인다. 다음 장에서는 원본 RAID 5 모듈과 수정된 RAID 5 모듈의 실제 실험치를 비교한다.

4. 성능 실험

호스트로의 요구를 처리하기 위해서 한 개의 쓰레드만을 사용하지만 디스크의 처리 수행을 완료할 때까지 기다리지 않고 SCSI 요구를 처리한다. 디스크 완료 후의 처리는 비동기적으로 이루어지므로, 많은 쓰레드를 사용하는 방법에 비해서 필요한 스케쥴링 오버헤드 및 자원이 적음에도 불구하고, 모든 호스트의 요구를 병렬로 처리할 수 있어 높은 처리율을 보인다. 또한 RAID 5 모듈내의 읽기 캐쉬를 제거하여 기존의 리눅스 RAID 5 모듈의 성능을 개선하였다.



(그림 6) 처리율 실험측정 비교



(그림 7) 응답시간 측정결과 비교

(그림 6)과 (그림 7)은 본 시스템의 개선한 RAID 5, 기존의 RAID 5, 그리고 타 장비의 읽기/쓰기 성능 측정한 것을 보여준다. RAID 5의 스트라이프 크기는 128kB로 정하였고, 하드디스크는 5개의 Seagate ST318273FC를 사용하였다. 이것은 18GB 용량의 7200rpm 회전율을 가진다. 호스트의 어댑터 카드는 QLogic의 QLA2300F를 사용하였다. 호스트 컴퓨터로는 펜티엄 III 733MHz의 Windows XP를 사용하였고, 벤치마크 도구로는 iometer 0.99를 사용하였다. 이 벤치마크 도구에서 지연된 입출력(pendded io)의 수를 4개로 정하였다. 이 실험에서 비교되는 타 장비는 EMC FC5300 모델이다. 이것은 30개의 1Gbps 파이버 채널 디스크를 설치할 수 있다. 또한 이 EMC 장비에는 본 시스템에서 사용한 하드디스크보다 성능이 좋은 10000rpm의 ST318203FC 5개를 사용하였다.

(그림 6)은 각 경우에 대한 대역폭(처리율)을 보여주고, (그림 7)은 각 경우에 대한 응답시간을 보여준다. 가로축은 실험

에서 사용한 블록의 크기를 의미한다. "읽기(수정)"과 "쓰기(수정)"은 개선한 RAID 5의 읽기와 쓰기 성능을 나타낸다. "읽기(원본)"과 "쓰기(원본)"은 원래의 리눅스 RAID 5의 읽기와 쓰기 성능을 나타낸다. "읽기(EMC)"와 "쓰기(EMC)"는 타 기종 EMC FC5300의 읽기와 쓰기 성능을 나타낸다. 이상의 것은 순차적인 읽기/쓰기의 성능이다. "무작위(수정)"과 "무작위(원본)"은 각각 개선한 RAID 5와 원래의 것에 대해서 50%쓰기와 50%읽기를 무작위 위치의 블록에서 읽을 때의 성능이다.

(그림 6)에서 볼 수 있듯이 개선한 본 시스템은 모든 경우에 대해서 가장 높은 성능을 보인다. 그런데 쓰기의 경우는 개선한 것과 원래의 것의 성능에는 전혀 변화가 없지만, 읽기는 2배 이상의 높은 성능 향상을 보인다. 왜냐하면 기존의 RAID 5 모듈은 메모리 복사 오버헤드로 인하여 이미 제어기의 성능이 포화되어 큰 블록 크기에도 성능의 향상이 거의 없기 때문이다. 반면, 개선한 것은 블록 크기에 비례해서 성능이 증가하는 것을 볼 수 있다.

기존의 RAID 5는 작은 쓰기를 위한 읽기 캐쉬를 하므로 읽기와 쓰기를 무작위로 하였을 경우에 캐쉬 히트가 발생한다. 이를 위한 실험으로서 50%쓰기와 50%읽기를 무작위 위치의 블록에서 읽을 때의 성능을 시험해 보았다. 그러나 개선한 것이 기존의 성능보다 높다는 것을 (그림 6)의 실험에서 볼 수 있다. 그 이유는 캐쉬 히트로 인한 성능 개선보다 메모리 복사 오버헤드를 없애어 생기는 성능 향상이 더 크기 때문이다.

(그림 7)은 (그림 6)과 같은 경우에 대해서 응답시간을 측정한 결과를 보여준다. 순차적 읽기/쓰기와 임의의 읽기/쓰기 모두에서 개선한 리눅스 RAID 5 모듈이 기존의 것보다 빠른 읽기 응답속도를 갖는다. 또한 본 시스템은 더 높은 성능의 하드디스크를 탑재한 타 기종 EMC FC5300보다 더 높은 성능과 빠른 응답시간을 보인다.

5. 결 론

본 논문은 내장형 리눅스를 이용한 RAID 시스템을 제시하였다. 파이버 채널 하드디스크와 호스트와의 정합을 위하여 파이버 채널 제어기를 포함하는 내장형 하드웨어를 설계 및 제작하였다. 이 시스템은 호스트에게 2Gbps 파이버 채널로 연결되어 RAID기능을 가지는 SCSI 블록 장치로 제공된다. 리눅스가 제공하는 Multi-device 모듈을 이용하여 RAID 0, RAID 1, RAID 5와 논리 볼륨 관리자 기능을 제공하고, 대용량의 데이터 캐쉬 기능 지원으로써 높은 성능을 제공한다. 파이버 채널 목표 모드 디바이스 드라이버, 목표 모드 SCSI 모듈, 데이터 캐쉬 모듈을 구현하였다.

리눅스 Multi-device의 RAID 5 모듈에서 작은 쓰기를 위한 읽기 캐쉬를 제거함으로써 메모리 복사 오버헤드를 제거하여 성능을 개선하였다. 수정된 RAID 5 모듈은 기존의 리눅스 RAID 5 모듈보다 읽기 및 임의의 입출력 부하 성능이 모든 경우에 대해서 우수함을 보인다. 또한 비슷한 사양의 타 기종과 성능을 비교하였을 때에도 본 시스템은 더 높은 대역폭과 빠른 응답속도를 보인다.

앞으로 RAID 5 모듈의 쓰기 캐쉬를 버퍼 캐쉬와 연동하여 쓰기 성능을 높여야하는 과제와 최근의 분산 RAID[15] 기술을 적용하여 고차원적인 상용 시스템을 개발하는 과제가 남아있다.

참 고 문 헌

[1] Clit Jurgens, "Fibre Channel : A Connection to the Future," IEEE Computer, Vol.28, No.8, pp.82-90, August, 1995.

[2] 이진희, "Next Generation Intelligent SAN Switches," 정보처리학회 자료저장시스템 학술대회, pp.66-91, 2002.

[3] D. A. Patterson, P. Chen, and R. H. Katz. "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," In 34th IEEE Computer Society International Conference, pp.112-117, 1989.

[4] M. Icaza, I. Molnar, and G. Oxman, "Kernel Korner : The New Linux RAID Code," Linux Journal, Vol.1997, Issue 44es, Dec., 1997.

[5] J. Edwards, A. Malmin, and R. Shaker, "RAID-1, Part 2," Linux HOWTO, August, 2002.

[6] J. Lombardo. "Embedded Linux," New Riders Publishing, 2001.

[7] 김정녀, 정교일, 이철훈, "리눅스 시스템의 버퍼 오버플로우 공격 대응 기법", 정보처리학회논문지A, 제8-A권 제4호, pp.385-390, 2001.

[8] A. Varma, and Q. Jacobson, "Destage Algorithms for Disk Arrays with Non-volatile Caches," ACM SIGARCH, Vol.23, May, 1995.

[9] "I/O ProcessorBased on Intel XScale Technology," http://developer.intel.com/.

[10] "Dolphin PCI-SCI Adapter Card," http://www.dolphinics.com/.

[11] "GNU RedBoot," http://www.redhat.com/.

[12] J. Epplin, "A developer's review of LynuxWorks' BlueCat Linux SDK," LynuxWorks, Dec., 2001.

[13] G Kroah-Hartman, "Kernel Korner : Hot Plug," Linux Journal, Issue 96, April, 2002.

[14] "Fibre Channel Host Bus Adapter QLA2300," http://www.qlogic.com/.

[15] K. Hwang, H. Jin, and R. Ho, "RAID-x : a new distributed disk array for I/O-centric cluster computing," International Symposium on High-performance Distributed Computer, pp.279-86, 2000.

백 승 훈

e-mail : shbaek@etri.re.kr

1997년 경북대학교 전자공학과(학사)

1999년 한국과학기술원 전기및전자공학과 (석사)

1999년~현재 한국전자통신연구원(컴퓨터 시스템연구부 연구원)

관심분야 : 자료 저장 시스템, 스트림 서버, 리눅스, 운영체제 등

박 종 원

e-mail : cwpark@etri.re.kr

1981년 한양대학교 전자통신공학과(학사)

1983년 한양대학교 전자통신공학과(석사)

2002년 한양대학교 전자통신공학과(박사)

1984년~현재 한국전자통신연구원 입출력 시스템연구팀장/책임연구원

관심분야 : RAID 시스템, 컴퓨터 네트워크, 시스템 아키텍춰 데이터 통신 등