NAS를 위한 신뢰성과 확장성이 있는 저장 장치 시스템

이 태 근† · 강 용 혁†† · 엄 영 익†††

요 약

대용량, 고속의 멀티미디어 서비스가 일반화됨에 따라 컨텐츠를 저장하는 서버의 저장 장치 또한 대용량화 되고 있다. 그러나 서버를 구성하는 다른 하드웨어 구성 요소에 비해 상대적으로 느린 저장 장치의 입출력 속도와 물리적 또는 논리적인 오류는 시스템 전반의 성능을 저하시키는 원인이 되고 있다. 또한 컨텐츠의 지속적인 증가로 인한 유연한 확장성이 필요하게 되었다. 본 논문에서는 이러한 문제점을 해결하기 위해 소프트웨어 RAID를 사용하여 저장 장치의 성능을 향상시키고 신뢰성을 높였으며, LVM을 사용하여 저장 장치에 확장성을 부여하였다. 리눅스 커널 2.4.x에서 제공하는 이러한 기능들을 사용하여 신뢰성과 확장성이 있는 저장 장치를 구현하고 성능을 평가하였다.

Reliable and Flexible Storage System for NAS Environments

Taekeun Lee† · Yong-Hyeog Kang†† · Young Ik Eom†††

ABSTRACT

The rapid growth of high speed multimedia services demands the large capacity of the server's storage that maintains the multimedia contents. Among the server's devices, the low I/O speed and physical or logical failure of storage device decrease the total performance of system. The continuous increase of multimedia contents require the flexibility of storage capacity. In order to solve the these problems, we propose the uses of software RAID and LVM techniques that provide the performance improvement and reliability of storage device and the flexibility of storage device respectively. In the LINUX 2.4 kernel, we implemented the reliable and flexible storage device and evaluated the performance of it.

키워드 : RAID(Redundant Array of Independent Disks), 논리적 볼륨 관리자(LVM : Logical Volume Manager), 네트워크 저장 장치(NAS : Network Attached Storage), 리눅스(LINUX)

1. 서 론

최근의 정보량의 증가 추세는 폭발적이라고 할 만큼 급격히 늘어나고 있다. 기존 텍스트 데이터와 같은 기본적인 형태의 데이터는 물론이고 오디오, 비디오, 이미지 등과 같은 대용량의 멀티미디어 데이터가 주류를 이루고 있다. 또한 네트워크의 발달로 인하여 대용량 멀티미디어 서비스가 확산되고 있다. 이러한 추세에 발맞추어 우리가 일반적으로 사용하는 PC에도 이미 수십 기가바이트 대의 하드디스크가 장착되고 있듯이 저장 장치의 용량 또한 급격히 증가하는 추세이다. 저장 장치가 대용량화되어 많은 양의 데이터를 저장할 수 있지만 저장 장치의 오류로 인해 한순간에 잃을 수 있는 데이터의 양도 그만큼 더 커졌으므로 신뢰성에 대한 요구가 커지고 있으며 대용량화를 필요로 하는 저장 장치를 보다 쉽게 확장하기 위한 요구 또한 커지고 있다. 따라서 서버를 사용하는 클라이언트들은 보다 신뢰성이 있으며 빠른 성능의 저장 장치를 필요로 하게 되었는데 이것에 대응하는 기술이 RAID(Redundant Array of Independent Disks)이다. 또한 기존 시스템의 환경을 그대로 유지하면서 급변하는 저장 장치 용량에 대응하는 기술이 LVM(Logical Volume Manager)이다.

최근에는 네트워크에 참여하는 구성원들의 데이터 공유를 목적으로 일반적인 서버를 사용하는 대신 저장 장치 기능을 전담하는 독립적인 네트워크 저장 장치(NAS : Network Attached Storage)를 사용하는 추세이다. 이러한 네트워크 저장 장치는 네트워크 상의 이 기종 클라이언트에게 효율적으로 파일 서비스를 제공할 수 있다.

본 논문에서는 이러한 요구 사항들을 수용하기 위해 소프트웨어 RAID와 LVM을 적용하여 신뢰성과 확장성이 있는 NAS를 위한 저장 장치를 구현하고 성능을 평가하였다. 본 논문의 2장에서는 저장 장치 구현과 관련된 연구에 대해 기술하고, 3장에서는 저장 장치의 설계 및 구현에 대해

※ 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2002-042-A00020)
† 정 회 원 : 성균관대학교 대학원 전기전자 및 컴퓨터공학과
†† 준 회 원 : 극동대학교 경영학부 교수
††† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2003년 5월 6일, 심사완료 : 2003년 8월 20일

기술한다. 4장에서는 저장 장치에 대한 성능을 평가하며 마지막으로 5장에서는 결론에 대해 기술한다.

2. 관련 연구

2.1 NAS(Network Attached Storage)

NAS란 파일서버 기능이 내장된 네트워크 저장 장치이다. 일반적인 저장 장치가 붙어 있는 서버의 경우 응용 서비스와 파일 서비스를 동시에 수행하므로 입출력, CPU, 네트워크에 병목 현상을 발생시키는 문제점을 갖는다. 뿐만 아니라 확장에 한계를 가지며 서버가 다운될 경우 모든 서비스가 중단되는 단점이 있다. 따라서 NAS는 응용과 데이터를 분리하여 독립적으로 파일서버 기능만을 처리하도록 만든 네트워크 저장 장치이다.

NAS는 기존의 Ethernet이나 TCP/IP 같은 네트워크 프로토콜을 사용하여 독립적으로 네트워크에 참여하며 응용 서버와는 별도로 존재한다. 따라서 NAS는 이 기종간의 파일 공유를 지원해야 하며 네트워크 및 디스크 성능, 신뢰성, 확장성 등이 최적화 되어야 한다[1, 2].

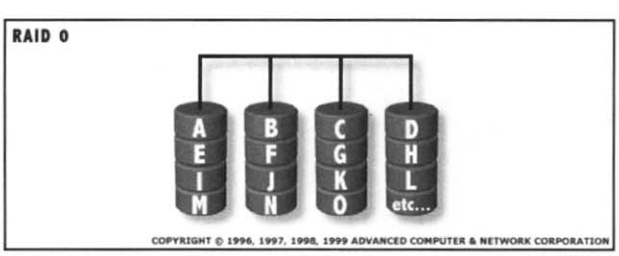
2.2 RAID(Redundant Array of Independent Disks)

RAID는 여러 개의 디스크를 사용하여 데이터를 저장하는 방법으로써 성능과 신뢰성을 향상시킬 수 있다. RAID의 구현 레벨에는 현재 0, 1, 2, 3, 4, 5, 6, 7, 53, 0+1이 있으며 리눅스 커널에서는 가장 일반적으로 사용되는 레벨 0, 1, 4, 5를 구현하고 있다. 본 장에서는 RAID 레벨 0, 1, 5에 대해서 알아본다.

2.2.1 RAID 구현 레벨

(1) RAID 레벨 0(striped disk array without fault tolerance)

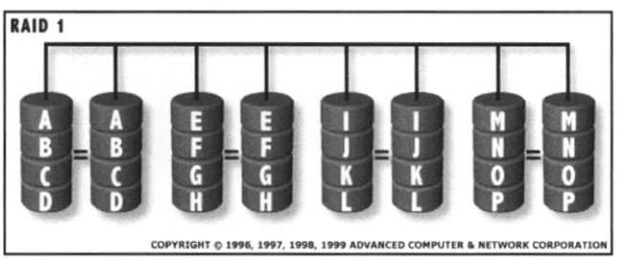
RAID 레벨 0의 개념도는 (그림 1)에서 보이는 바와 같다. RAID 레벨 0은 2개 이상의 디스크로 구성된다. 데이터는 분산(Stripping)되어 여러 개의 디스크에 기록된다. 따라서 여러 개의 디스크에 동시에 입출력이 가능하기 때문에 입출력 성능은 RAID를 구성하는 디스크의 개수에 비례하게 증가하며, 전체 디스크 용량을 모두 사용할 수 있다는 장점이 있다. 하지만 디스크 장애시 데이터를 복원하는 어떠한 방법도 적용되어 있지 않다는 단점이 있다. 따라서 진정한 의미의 RAID라고는 할 수 없다[3, 4].



(그림 1) RAID 레벨 0 개념도

(2) RAID 레벨 1(mirroring and duplexing)

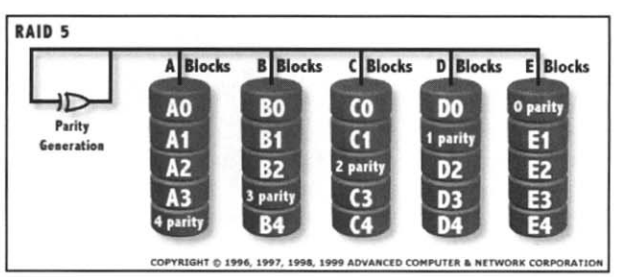
RAID 레벨 1의 개념도는 (그림 2)에서 보이는 바와 같다. RAID 레벨 1은 2개 이상의 짝수개의 디스크로 구성된다. 데이터를 디스크에 기록할 때 쌍을 이루는 디스크에 동일한 데이터를 기록한다. 따라서 디스크 장애 시 복제된 다른 디스크를 사용함으로서 데이터를 간단히 복원할 수 있다. 디스크 입력 성능은 단일 디스크를 사용할 때와 같으며, 디스크 출력 성능은 쌍을 이루는 디스크가 동시에 동작하므로 단일 디스크를 사용할 때 보다 2배 정도 좋은 성능을 가진다. 하지만 전체 디스크 용량의 절반만 사용 가능하고, 구성비용이 가장 많이 든다는 단점이 있다[3, 4].



(그림 2) RAID 레벨 1 개념도

(3) RAID 레벨 5(independent data disks with distributed parity blocks)

RAID 레벨 5의 개념도는 (그림 3)에서 보이는 바와 같다. RAID 레벨 5는 3개 이상의 디스크로 구성된다. 데이터의 복원 방식은 패리티 방식을 사용한다. 데이터는 블록 단위로 기록되며, 패리티도 데이터와 마찬가지로 데이터용 디스크에 분산되어 기록된다. RAID 레벨 4에서는 패리티를 하나의 디스크에 저장하는 방식을 사용하는데 이것은 패리티 정보가 하나의 디스크에 집중되어 병목 현상을 일으키는 문제점을 갖는다. RAID 레벨 5는 이러한 패리티 정보를 RAID를 구성하는 디스크들에 고르게 분산하여 RAID 레벨 4가 갖는 병목현상을 제거하였다. RAID 레벨 5는 RAID 레벨 4와 마찬가지로 하나의 디스크의 오류에 대해서만 안전하며 2개 이상의 디스크가 동시에 오류를 일으킬 경우 모든 데이터를 잃을 수 있다[3, 4].



(그림 3) RAID 레벨 5 개념도

2.2.2 하드웨어 RAID와 소프트웨어 RAID

RAID를 구현하는 방법에는 하드웨어 RAID와 소프트웨어 RAID가 있다. 하드웨어 RAID는 전용의 프로세서와 메모리 등을 사용하여 구현되므로 빠른 속도를 내긴 하지만, 구현을 위해 하드웨어 구성 요소가 필요하기 때문에 비용이 많이 든다. 또 하드웨어 RAID는 구현할 때 사용한 하드웨어 구성 요소들에 의해 성능의 상한이 정해지므로 오래된 하드웨어 RAID의 경우 시스템을 구성하는 다른 하드웨어 구성 요소들과의 성능 불균형으로 인해 시스템 전반의 속도를 저하시킬 수 있다. 때로는 소프트웨어 RAID를 사용하는 것보다 더 느릴수도 있다.

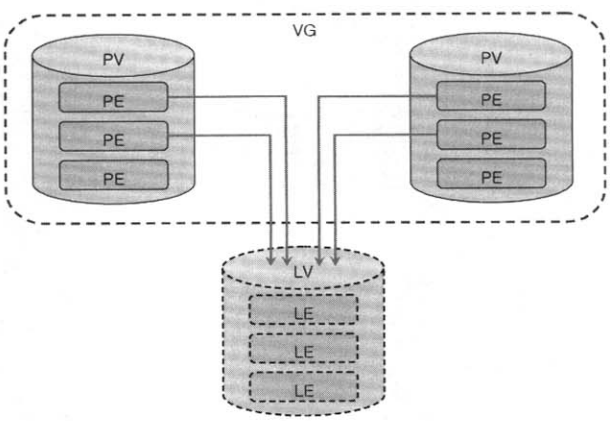
소프트웨어 RAID는 하드웨어 RAID와는 반대로 전용의 프로세서나 메모리 등의 하드웨어를 사용하지 않고 RAID와 관련된 연산을 커널에서 처리하도록 구현한 것이다. 따라서 소프트웨어 RAID는 하드웨어 RAID에 비해 구현비용이 저렴하지만 RAID와 관련된 연산을 처리하기 위하여 CPU 및 메모리 등의 시스템 자원을 그만큼 더 소비한다는 단점이 있다. 또 소프트웨어 RAID는 커널에 종속적인 기능이므로 이 기종 시스템으로의 이식성이 떨어지는 단점이 있다. 하지만 RAID를 구성하는 하드웨어 구성 요소들의 성능에 종속적인 하드웨어 RAID와는 반대로 CPU의 성능에 따라 RAID의 성능이 가변적인 유연성을 가지고 있다. 또 일반적으로 특정한 하드디스크에 대해서만 호환성을 제공하는 하드웨어 RAID와는 반대로 커널이 인식하는 하드디스크라면 RAID를 구성함에 있어서 호환성을 걱정하지 않아도 되는 장점이 있다.

따라서 최근에는 고성능 CPU가 일반화 되고 있으며 빠른 기술 발달로 인해 하드웨어 RAID의 장점이 퇴색하고 있는 경향이므로 적은 비용이 들며 고성능이면서도 신뢰성이 있고 호환성과 성능의 유연성에서 자유로운 소프트웨어 RAID를 사용하는 것이 바람직한 선택이다. 리눅스 커널에서는 이러한 소프트웨어 RAID를 매우 잘 지원하고 있다[1, 5].

2.3 LVM (Logical Volume Manager)

LVM이란 복수의 디스크 또는 파티션을 종합해 하나의 논리적인 영역으로 취급하는 것이다. LVM으로 구성된 논리적인 영역은 커널에서는 하나의 물리적인 디스크 장치처럼 보이기 때문에 디스크와 파티션 구성에 유연성을 갖도록 할 수 있다. RAID의 경우 여러 개의 디스크를 이용하여 성능과 신뢰성을 향상시키는데 목적이 있지만 LVM은 시스템 운영시 동적으로 디스크 또는 파티션 구성을 바꿀 수 있는 확장성을 제공하는데 목적이 있다.

LVM으로 구성된 논리적인 볼륨은 작성한 후에라도 새로운 파티션을 추가 또는 삭제할 수 있다. 따라서 시스템 운영 중에 디렉터리의 크기를 변경할 수 있는 유연성을 제공한다. LVM의 개념도는 (그림 4)에서 보이는 바와 같다. 디스크 또는 파티션을 LVM 전용 파티션으로 만든 것을 PV(Physical Volume)라고 한다. PE(Physical Extent)는 PV를 구성하는 기본 단위로서 기본 크기는 4MB이다. VG(Volume Group)는 복수개의 PV로 구성된 하나의 영역으로 가상의 하드디스크라고 할 수 있다. LV(Logical Volume)는 VG내의 PE 몇 개를 할당하여 만든 가상 파티션에 해당하는 것으로 실제 파일시스템이 작성되게 된다. LE(Logical Extent)는 LV에 할당된 VG내의 PE를 지칭한다. (그림 4)에서 보이는 바와 같이 LV에 PE를 얼마만큼 할당하느냐에 따라 LV의 크기를 조정할 수 있는 유연성을 갖게 되는 것이다[6-8].



(그림 4) LVM 개념도

3. 소프트웨어 RAID 레벨 5와 LVM이 적용된 저장 장치

3.1 저장 장치의 구현

본 논문에서는 소프트웨어 RAID 레벨 5로 구현된 저장 장치에 LVM을 적용하여 성능과 신뢰성 향상, 그리고 관리의 유연성을 제공하는 시스템을 구축하였다. 저장 장치의 구현에 사용된 하드웨어는 <표 1>에서 보이는 바와 같으며 소프트웨어는 <표 2>에서 보이는 바와 같다.

<표 1> 구현에 사용된 하드웨어 및 사양

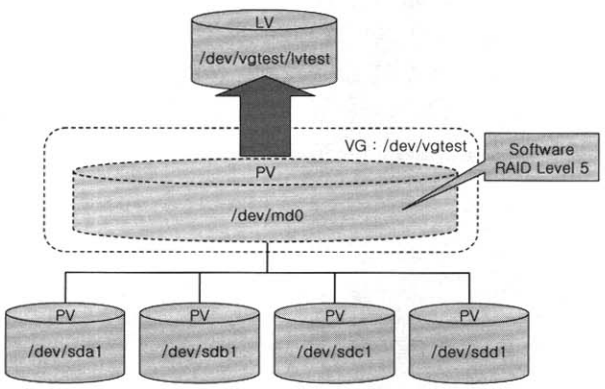
종 류	사 양
CPU	AMD Athlon 1 Ghz
Main Memory	256 MB
SCSI Interface	Ultra 2 type
H.D.D.	Quantum Atlas V 18.3 GB (4 EA)

LVM 기능을 사용하기 위해서는 리눅스 커널에 LVM 패치를 적용한 후 커널을 컴파일을 해야 한다. 이때 소프트웨

어 RAID 및 LVM을 커널 모듈로 컴파일 하거나 커널에 포함시켜 컴파일 해야 한다[7].

〈표 2〉 구현에 사용된 소프트웨어 및 버전

종 류	버 전
운영체제	와우리눅스 7.1
리눅스 커널	2.4.7
LVM 소프트웨어	1.0.3



(그림 5) 소프트웨어 RAID 레벨 5와 LVM이적용된 저장 장치

본 논문에서 구현한 저장 장치 시스템은 (그림 5)에서 보이는 바와 같다. 이 저장 장치 시스템은 아래와 같은 순서를 거쳐서 구현된다.

① 디스크 4개를 소프트웨어 RAID 레벨5를 적용하여 하나의 저장 장치로 구성
  • 리눅스에서 /dev/mdx로 인식[5]
② ①에서 구성한 저장 장치를 PV로 만듦
③ ②에서 구성한 PV(/dev/mdx)를 VG에 포함 시킴
  • 임의로 VG의 이름을 vgtest로 설정
④ VG로부터 PE를 할당하여 LV를 생성
  • 임의로 LV의 이름을 lvtest로 설정
  • /dev/vgtest/lvtest 블록 디바이스 생성
⑤ /dev/vgtest/lvtest에 파일시스템을 작성하고 마운트 하여 사용

현재 LVM으로 작성된 LV에 작성할 수 있는 파일시스템은 ext2, reiserfs와 xfs가 있다. reiserfs와 xfs같은 저널링 파일시스템을 사용하면 정전과 같은 상황으로부터 빠르게 파일시스템 복구가 가능한 고 가용성 기능도 추가할 수 있다. 그러나 본 논문에서는 리눅스에서 가장 일반적으로 사용되는 ext2 파일시스템을 사용하여 저장 장치를 구현하였다[7, 9].

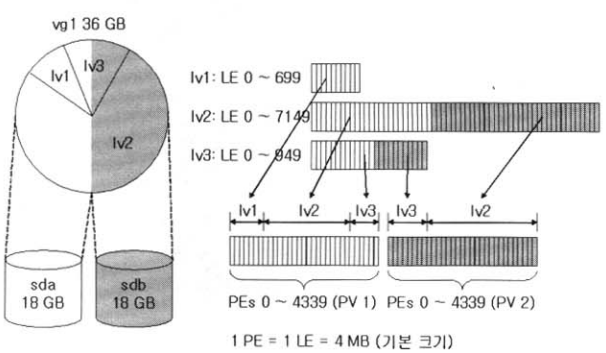
3.2 LVM 동작 과정

각 PV들은 동일한 크기의 PE로 나누어지게 된다. PE의 크기는 관리자가 설정하기에 따라서 유동적이지만 VG에 속하게 되는 모든 PE의 크기는 같아야 한다. 각 PV들은 0~PV를 구성하는 전체 PE의 개수(PV의 크기/PE의 크기)로 번호가 할당된 PE들로 구성되어지며 이 번호는 각 PV에 대하여 고유하다.

각 PV들은 시작 부분에 VGDA(Volume Group Descriptor Area)라고 하는 LVM 설정 메타데이터를 가지고 있다. 이 VGDA에는 PV descriptor, VG descriptor, LV descriptors, PE descriptors에 대한 정보가 저장된다. 여기에 저장되지 않은 LE descriptors에 대한 정보는 LVM이 활성화 되었을 때 할당된 PE로부터 유추하게 된다. 이와 같이 VGDA에는 LVM을 구성하는 중요한 정보들이 저장되므로 LVM이 적용된 시스템을 백업할 때 반드시 백업 목록에 VGDA를 포함 시켜야 할 것이다. 이러한 VGDA는 커널이 초기화 될 때 버퍼 캐시로 읽어 들여 VG들과 LV들을 활성화 하는데 사용된다. 즉, VGDA로부터 LV가 실제 어떤 물리적인 저장 장치와 연관되는지에 관련된 사상(mapping) 테이블을 구성하게 되고 이러한 사상 테이블은 LV에 대한 입출력 연산이 일어날 때 참조 된다.

모든 LV는 PV가 동일한 크기의 PE들로 나누어진 것처럼 PE와 같은 크기의 LE들로 나누어지게 된다. 각 LV들은 0~LV를 구성하는 전체 LE의 개수(LV의 크기/LE의 크기)로 번호가 할당된 LE들로 구성되어지며 이 번호는 각 LV에 대하여 고유하다. 모든 LE는 PV상의 PE와 정확히 1 : 1로 사상된다(LE : (PV : PE)).

응용 프로그램에서 어떤 LV상에 작성된 파일시스템에 대한 입출력 요청이 발생하면 입출력 대상이 되는 LE들의 번호를 바탕으로 실제 입출력이 일어나게 될 PV들과 PE들을 사상 테이블을 참조하여 알아낸다. 따라서 실제 입출력이 일어나야할 물리적인 장치의 정확한 위치가 파악 되었으므로 기존 물리적인 장치의 입출력 방식과 같은 과정으로 입출력이 발생하게 된다. 이와 같은 LVM의 사상 상태는 (그림 6)에서 보이는 바와 같다.



(그림 6) LVM의 사상 상태

예를 들어 응용 프로그램에서 lv3의 254,123byte에 접근이 발생했다고 가정하자. (그림 6)에서 보이는 바와 같은

사상 상태에서는 해당 LE의 번호는 62가 된다(62 * 4MB = 253,952). lv3의 62번 LE는 사상 정보에 의해서 PV1로 사상되고 PE의 번호는 lv1의 PE들의 개수 + lv2의 PE들의 개수 + 62로 결정된다[8].

3.3 LVM 사용으로 인한 오버헤드

3.2절에서 살펴보았듯이 커널 내부의 LVM 층은 저장 장치의 논리적인 저장 위치와 물리적인 저장 위치의 사상 테이블을 유지한다. 이것은 응용 프로그램에서 LVM으로 구성된 저장 장치에 읽기 또는 쓰기를 위해서 접근을 할 때 커널 내부의 LVM 층에 접근하여 사상 테이블로부터 논리적인 저장 위치를 물리적인 저장 위치로 변환해야 한다는 것을 의미한다. 따라서 LVM으로 구성되지 않은 일반적인 저장 장치의 접근에 비하여 LVM으로 구성된 저장 장치로의 접근은 단 1회의 메모리 참조 연산이 추가될 뿐이므로 오버헤드는 크지 않다. 왜냐하면 고성능 CPU가 일반화 되어 있는 현재와 같은 컴퓨팅 환경에서는 기계적인 저장 장치에 접근하는데 소모되는 시간에 비하여 추가로 발생하는 메모리에 대한 1회의 참조 연산 시간은 무시할 수 있을 만큼 작은 시간이기 때문이다[8].

3.4 RAID 장치 추가를 통한 저장 장치의 확장

3.1절에서와 마찬가지 방법으로 새로운 소프트웨어 RAID 장치를 만든다. 기존의 RAID 장치는 이미 VG(vgtest)에 포함되어 있으므로 이러한 VG에 새로 작성한 RAID 장치를 포함시키면 VG에 포함된 PE의 개수가 늘어나 LV에 더 많은 PE를 할당할 수 있게 된다. 간단히 VG에 새로운 PV 추가하는 것만으로 LV의 사이즈를 확장할 수 있다. LV를 확장한 후에는 각 파일시스템에 해당하는 크기 재조정 도구를 이용하여 파일시스템의 크기를 조정해 주고 마운트하여 사용하면 된다[7].

4. 성능 평가

본 장에서는 3장에서 구현한 저장 장치의 성능 평가를 수행하였다. 4.1절에서는 호스트 시스템에 구성된 저장 장치 본연의 성능을 평가 하였으며 4.2절에서는 NFS와 저장 장치가 연동하여 NAS를 구성하였을 때의 성능을 평가하여 NAS에 적합한지 살펴보았다.

성능 평가 도구로는 bonnie++ 1.01d를 사용하였다. bonnie++는 POSIX 표준 API를 사용하여 파일시스템의 입출력속도와 CPU 사용률을 측정한다. NAS는 주로 대용량의 파일 서비스를 목적으로 하므로 본 논문에서는 bonnie++가 제공하는 성능 평가 항목 중에 블록 쓰기 및 읽기, 임의의 파일 찾기(Random Seeks), 그리고 각 평가 항목에 대한 CPU 사용률을 측정하여 성능 평가를 수행하였다[10, 11].

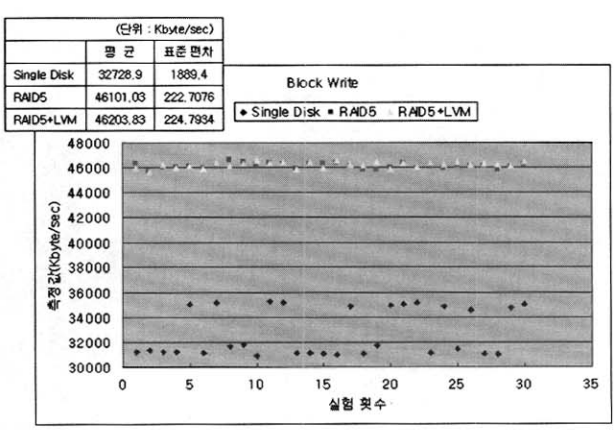
성능 평가는 다음과 같이 3가지의 경우로 나누어 평가하였다.

① 디스크 1개를 사용한 경우(Single Disk)
② 디스크 4개로 소프트웨어 RAID 레벨 5를 구성한 경우(RAID5)
③ RAID5에 LVM을 추가한 경우(RAID5 + LVM)

각각의 경우 30회씩 bonnie++를 사용하여 측정한 후 결과를 정리하였다. bonnie++는 성능 평가를 위한 데이터를 생성할 때 메모리로 인한 캐시 효과를 제거하기 위해서 시스템에 설치된 메모리 용량의 2배 이상을 생성하도록 권장하고 있다[11]. 따라서 4.1절의 경우 호스트 시스템에 설치된 메모리 용량인 256MB의 2배가 넘는 600MB의 파일을 생성하고 삭제하도록 하였다. 역시 마찬가지로 4.2절의 경우 쓰기 및 읽기를 요청하는 NFS 클라이언트 시스템에 설치된 메모리 용량인 256MB의 2배가 넘는 600MB의 파일을 생성하고 삭제하도록 하였다.

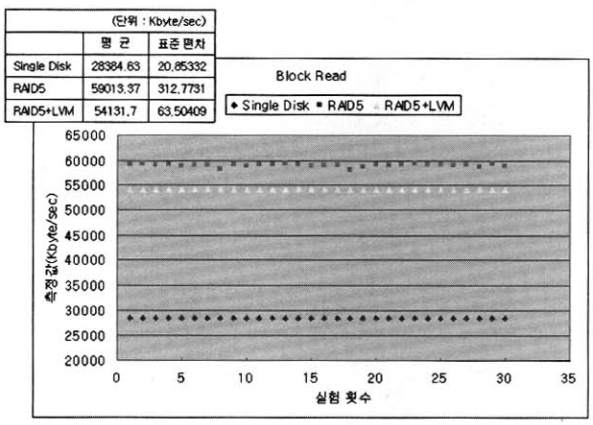
4.1 로컬 시스템 상의 저장 장치의 성능 평가

블록 쓰기에 대한 성능 평가 결과는 (그림 7)에서 보이는 바와 같다. 여기서 주목할 점은 RAID5 + LVM의 경우 RAID5보다 한 단계 더 소프트웨어 층을 거침에도 불구하고 성능 차이가 거의 없다는 점이다. 실제로 그림 좌측 상단의 RAID5와 RAID5 + LVM 두 측정값을 비교해 보면 두 값의 차이가 오차범위에 속하므로 같은 성능을 가지고 있다고 보아도 무방하다. 또 (그림 7)에서 보이듯이 Single Disk는 측정값의 편차가 매우 심한데 반해 RAID5와 RAID5 + LVM은 매우 안정된 결과를 보여주고 있다. 이것은 RAID로 인하여 얻을 수 있는 추가적인 장점으로 여러 개의 디스크가 동시에 동작하며 쓰기 작업을 수행하므로 대용량의 쓰기 작업을 안정적으로 수행할 수 있기 때문이다.



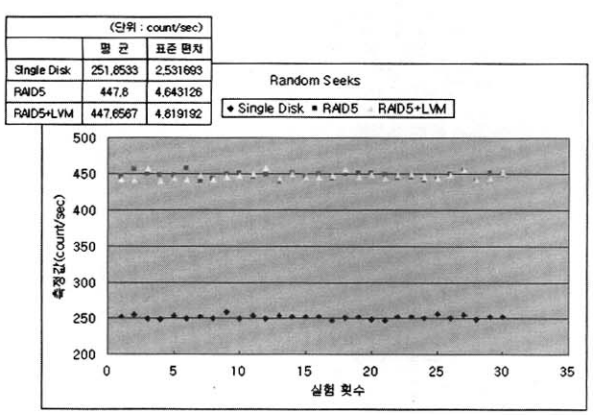
(그림 7) 블록 쓰기 결과

다음으로 블록 읽기에 대한 성능 평가 결과는 (그림 8)에서 보이는 바와 같다. 여기서 주목할 점은 RAID5와 RAID5+LVM이 Single Disk에 비해서 거의 2배에 가까운 성능 향상을 얻을 수 있었다는 것인데 대부분의 유닉스 파일시스템에서는 읽기와 쓰기의 비율이 65~80대 20~35 사이를 차지할 만큼 읽기의 비중이 높다는 점을 고려할 때 읽기 성능의 비약적인 향상은 전체 시스템의 성능 향상에 많은 도움을 줄 것이다[12]. 결과에서 보이듯이 블록 쓰기와는 달리 LVM의 사용으로 인한 약간의 성능 저하가 발생하고 있다.



(그림 8) 블록 읽기 결과

임의의 파일 찾기에 대한 성능 평가 결과는 (그림 9)에서 보이는 바와 같다. 이 성능 평가 결과는 블록 쓰기와 블록 읽기의 결과를 혼합한 듯한 모습을 보이고 있다. 블록 쓰기 결과와 유사하게 RAID5와 RAID5+LVM의 측정값이 거의 똑같은 값을 보이고 있다. 즉, LVM 사용으로 인한 성능 저하가 발생하지 않는다고 봐도 무방하다. 또 블록 읽기 결과와 유사하게 임의의 파일 찾기 결과에서도 Single Disk에 비해서 거의 2배에 가까운 성능 향상을 얻을 수 있었다.



(그림 9) 임의의 파일 찾기 결과

마지막으로 각 성능 평가에 항목에 대한 평균 CPU 사용률은 <표 3>에서 보이는 바와 같다. 쉽게 예상 할 수 있듯이 소프트웨어 RAID를 사용하기 때문에 관련 연산을 처리하기 위해 CPU 자원을 더 많이 소모 한다. 하지만 LVM이 추가 된다고 하더라도 CPU 사용률이 거의 증가 하지 않는 것으로 볼 때 LVM으로 인한 오버헤드는 거의 무시할만한 것이라는 것을 알 수 있다.

〈표 3〉 평균 CPU 사용률

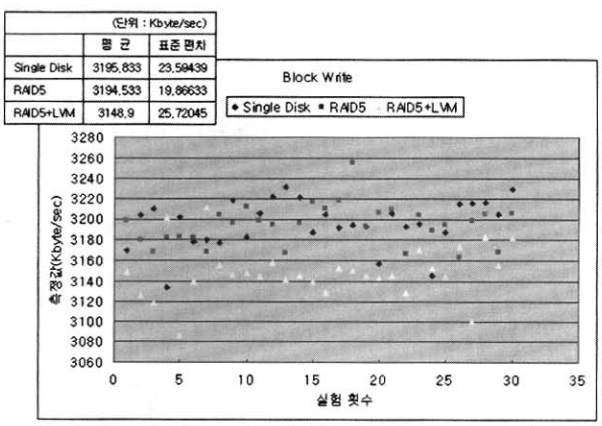
(단위 : %)

	Single Disk	RAID5	RAID5 + LVM
블록 쓰기	13.93	26.17	26.07
블록 읽기	11.27	30.01	25.53
임의의 파일 찾기	0.5	2.07	2.07

오히려 RAID5의 CPU 사용률이 더 높게 나온 이유는 RAID5 + LVM이 LVM에 관련된 연산을 처리하는데 사용한 시간만큼을 RAID5는 쓰기 및 읽기를 위해 더 사용했기 때문이다. 즉, RAID5 + LVM에 비하여 RAID5는 단위시간당 더 많은 데이터를 처리하기 위해 CPU 자원을 그만큼 더 소모했기 때문이다.

4.2 NFS와 연동한 저장 장치의 성능 평가

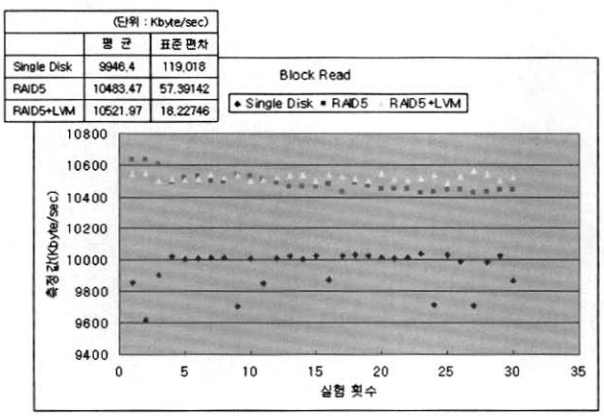
4.1절과는 달리 본 절에서는 3장에서 구현한 저장 장치가 NAS로 동작하였을 때의 성능을 평가 하였다. NAS로 동작하게 하기 위해서 저장 장치가 구현된 호스트 컴퓨터에 NFS 서버 기능을 추가하였다. NFS 클라이언트로는 Intel Pentium 3 1Ghz 시스템에 한컴리눅스 2.2를 설치하여 사용하였다. NFS 서버와 클라이언트의 구축에 사용된 네트워크 인터페이스는 Fast Ethernet으로 100Mbps의 대역폭을 가지고 있다. NFS 서버의 저장 장치를 NFS 클라이언트로 마운트하고 클라이언트에 설치된 bonnie++를 사용하여 4.1과 같은 방식으로 성능을 평가하였다.



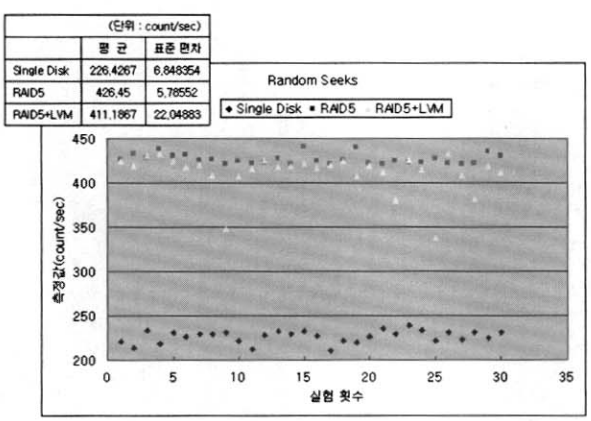
(그림 10) NFS 블록 쓰기 결과

블록 쓰기에 대한 성능 평가 결과는 (그림 10)에서 보이는 바와 같다. (그림 10)에서 보이듯이 블록 쓰기 결과는 어느 것이 우월한 성능을 보인다고 말할 수 없을 정도로 혼란한 결과를 보이고 있다. 단순히 평균값의 비교만으로는 Single Disk가 가장 성능이 좋다고 말할 수는 있지만 평가 대상간의 결과 차이가 200Kbyte 미만인 것을 볼 때 어느 것이 확실히 성능 우위에 있다고 말하기는 어렵다고 할 수 있다.

다음으로 블록 읽기에 대한 성능 평가 결과는 (그림 11)에서 보이는 바와 같다. (그림 11)에서 보이듯이 RAID의 사용으로 인해 4.1의 블록 읽기에서와 같이 읽기 성능의 향상을 확인 할 수 있었다. 그러나 4.1에서와 같이 큰 폭의 성능 향상이 보이지 않는 이유는 최고 100Mbps(12.5Mbyte/sec)의 대역폭만을 제공하는 네트워크의 성능 제약 때문이라고 할 수 있다. 네트워크의 성능 제약으로 인한 저장 장치의 성능 저하는 Gigabit Ethernet등의 더 빠른 네트워크 환경에서의 성능 평가를 통하여 확인할 수 있을 것이다.



(그림 11) NFS 블록 읽기 결과



(그림 12) NFS 임의의 파일 찾기

임의의 파일 찾기 성능 평가 결과는 (그림 12)에서 보이는 바와 같다. 여기서 주목할 것은 네트워크의 성능 제약이 존재함에도 불구하고 (그림 9)의 결과와 거의 같은 결과를 보여 주고 있다는 것이다. 이것은 앞에서 살펴보았던 NFS 환경에서의 블록 쓰기 및 읽기와는 대조적으로 임의의 파일 찾기는 실제 데이터의 네트워크를 통한 이동이 매우 적기 때문에 네트워크의 성능 제약을 거의 받지 않는다. 따라서 NAS를 사용하는 클라이언트의 경우 임의의 파일 찾기에 있어서는 로컬 시스템에서와 같은 빠른 응답 시간을 경험할 수 있을 것이다.

마지막으로 NFS 환경에서의 각 성능 평가 항목에 대한 평균 CPU 사용률은 <표 4>에서 보이는 바와 같다. 여기서의 CPU 사용률은 NFS 클라이언트의 CPU 사용률로서 실제 NAS를 사용하는 클라이언트 컴퓨터에 걸리는 부하량을 의미한다. <표 3>과는 반대로 RAID가 적용된 저장 장치를 가진 NFS 서버를 사용하는 NFS 클라이언트의 평균 CPU 사용률이 현저히 낮다. 이러한 결과는 NAS를 사용하는 클라이언트에 매우 유리한 것으로서 적은 CPU 사용률은 클라이언트의 작업에 부담을 덜어 줄 것이다.

〈표 4〉 NFS 클라이언트의 평균 CPU 사용률

(단위 : %)

	Single Disk	RAID5	RAID5+LVM
블록 쓰기	4	1	1
블록 읽기	23.4	1.83	1.87
임의의 파일 찾기	6.33	1.57	1.33

5. 결 론

본 논문에서는 저장 장치에 신뢰성과 확장성을 갖게 하기 위해 소프트웨어 RAID와 LVM을 사용하여 저장 장치를 구성하고 성능 평가를 통하여 효용성을 입증하였다. 성능 평가의 결과에서 보이듯이 소프트웨어 RAID 레벨 5를 사용하여 성능을 향상시킬 수 있었으며 LVM을 추가한다고 하더라도 성능 저하가 미약하므로 LVM을 추가하여 얻을 수 있는 장점을 고려했을 때 NAS에 LVM을 탑재하는 것이 적합한 선택임을 알 수 있다. 또한 NFS를 사용한 NAS 클라이언트의 성능 평가를 통해 이러한 구성이 NAS 클라이언트에 적합한 것임을 알 수 있었다. 물론 소프트웨어 RAID 레벨 5와 LVM을 탑재함으로써 NAS의 전체적인 CPU 사용률이 높아지는 단점이 발생하지만 NAS는 파일 서비스만을 제공하므로 추가적인 오버헤드가 필요하지 않다는 점을 고려해보면 이러한 단점이 시스템에 큰 영향을 미치지 않는다는 것을 알 수 있다.

따라서 RAID5를 사용하여 성능 및 신뢰성을 향상시킬 수 있고 LVM을 추가함으로써 저장 장치 확장이 유연해지므로 RAID5 + LVM기반의 NAS를 구현하는 것이 가장 적합한 선택임을 알 수 있다.

참 고 문 헌

[1] 손재기, 이형수, 민수영, 박창원, "임베디드 리눅스를 이용한 네트웍 저장장치 연구 개발", 제1회 자료저장 시스템 연구회 워크샵, pp.185-190, 2001.

[2] Garth A. Gibson and Rodney Van Meter, "Network Attached Storage Architecture," Communications of the ACM, Vol.43, No.II, pp.37-45, 2000.

[3] Peter M. Chen and Edward K. Lee and Garth A. Gibson and Randy H. Katz and David A. Petterson, "RAID : High-Performance, Reliable Secondary Storage," ACM Computing Surveys, Vol.26, No.2, pp.145-185, 1994.

[4] Advanced Computer & Network Corporation, "Get To Know RAID," http://www.acnc.com/04_01_00.html.

[5] Jakob Østergaard, "The Software-RAID HOWTO," http://www.linuxdoc.org.

[6] David Teigland, "Volume Manager in Linux," Sistina Software, Inc., 2001.

[7] AJ Lewis, "LVM HOWTO," http://www.sistina.com.

[8] Michael Hasenstein, "WHITEPAPER The Logical Volume Manager(LVM)," SuSE, Inc., 2001.

[9] Bar, Moshe, "Linux File Systems," McGraw Hill Osborne Media, 2001.

[10] Advanced Computer & Network Corporation, "Benchmarking Tools," http://www.acnc.com/04_02.html.

[11] Tim Bray and Russell Coker, "Bonnie++ Documentation," http://www.coker.com.au/bonnie++/readme.html.

[12] John K. Ousterhout and Hervé Da Costa and David Harrison and John A. Kunze and Mike Kupfer and James G. Thompson, "A Trace-driven Analysis of the 4.2 BSD UNIX file system," Proceedings of the tenth ACM symposium on Operating system principles, pp.15-24, 1985.

이 태 근

e-mail : tedlee@ece.skku.ac.kr

2002년 성균관대학교 전기전자 및 컴퓨터 공학부 학사

현재 성균관대학교 전기전자 및 컴퓨터 공학과 석사과정

관심분야 : Embedded Linux, 이동 컴퓨팅 시스템

강 용 혁

e-mail : yhkang1@ece.skku.ac.kr

1996년 성균관대학교 정보공학과 학사

1998년 성균관대학교 정보공학과 컴퓨터 공학전공 석사

2003년 성균관대학교 정보통신공학부 박사

현재 극동대학교 경영학부 전자상거래 전공 교수

관심분야 : 분산 시스템, 이동 컴퓨팅 시스템, Mobile Ad-hoc networks

엄 영 익

e-mail : yieom@ece.skku.ac.kr

1983년 서울대학교 계산통계학과 학사

1985년 서울대학교 대학원 전산과학전공 석사

1991년 서울대학교 대학원 전산과학전공 박사

2000년~2001년 Dept. of Info. and Comm. Science at UCI 방문교수

현재 성균관대학교 정보통신공학부 교수

관심분야 : 분산 시스템, 이동 컴퓨팅 시스템, 이동 에이전트, 시스템 소프트웨어