□특집□

리눅스 클러스터 환경에서의 분산 병렬 파일 시스템

서 대 화†

◈ 목 차 ◈

1. 서 론

최근 대용량의 분산 웹 서버와 VOD 서버 및 고성능의 컴퓨팅 파워와 대용량 자료 저장을 필요로 하는 서비스의 증가로 클러스터 시스템에 관한 연구가 활발하게 진행 중이다. 이러한 클러스터 시스템들의 주 응용 분야인 분산 웹 서비스나 VOD 서비스 그리고 대용량의 멀티미디어 자료 검색 서비스 등은 실시간에 대량의 파일 입출력 처리를 요구하고 있다.[1,4] 클러스터 환경에서 대용량의 파일 입출력을 지원해 주기 위해서 기존의 파일 시스템으로는 처리가 불가능하기 때문에 MPP나 슈퍼컴퓨터에서처럼 클러스터 환경에 적합한 분산 병렬 파일 시스템의 필요하다.

최근 저렴한 비용과 구성의 용이성 등으로 인하여 급속하게 연구 개발이 진행되고 있는 리눅스 클러스터에서도 분산 병렬 파일 시스템의 필요성이 대두되고 있다. 물론 기존 워크스테이션이나 중형급 컴퓨터를 이용한 클러스터 시스템을 공급하고 있던 일부 업체에서는 자체의 분산 병렬 파일 시스템을 제공하고 있다[11,16].

리눅스를 이용한 클러스터에 대한 연구가 아직 초기 단계에 있으므로 해서 분산 병렬 파일 시스템에 대한 연구도 초기 단계에 있지만, 리눅스 클러스터에 대한 연구 개발과 함께 리눅스 클러스터를 위한 새로운 병렬 파일 시스템이나 분산 파일 시스템을 개발하기 위한 연구도 전 세계적으로 활발히 진행 중에 있다. 현재 리눅스 클러스터를 위한 병렬 파일 시스템이나 분산 파일 시스템이 일부 대학에서 개발되고 있고, 또 일부 대학에나 연구소에서는 기존의 유닉스 시스템 환경을 위해서 개발된 여러 가지 분산 병렬 파일 시스템들을 리눅스 클러스터 환경에 이식하고 있다[2,7,10,12,17-18,20].

본 고에서는 최근 활발하게 진행되고 있는 리눅스 클러스터를 보다 효율적으로 사용할 수 있게 지원하는 병렬 분산 파일 시스템에 대한 각 대학이나 연구소 및 관련업체의 연구 개발 현황을 살펴보기로 한다. 먼저 2장에서 클러스터 컴퓨팅의 특징와 장단점에 대해서 알아보고, 3장에서 리눅스 운영체제에서의 파일 시스템에 대해서 기술한다. 그리고 4장에서 현재까지 연구 개발되거나 연구 중에 있는 리눅스를 위한 분산 병렬 파일 시스템에 대한 사례를 살펴보고, 결론을 맺기로 한다.

2. 클러스터 컴퓨팅 (Clustering Computing)

클러스터 컴퓨팅은 여러 대의 컴퓨터 시스템을 통합해 단일 시스템 이미지(single system image)를

† 정회원 : 경북대학교 전자전기공학부 교수

제공하여 보다 빠르고 지속적인 처리를 제공하는 시스템으로 정의되어진다. 우수한 클러스터 구조는 지원 기능의 범위가 자료 접근 수준에서의 가용성부터 광범위하게 분산된 컴퓨팅 자원들에 대해서도 응용 수준에서의 가용성을 제공해야만 된다.

클러스터는 워크스테이션들을 네트웍 환경에서 사용하는 것부터 리눅스 PC와 같은 컴퓨터 시스템들을 프로세서 노드로 사용한 실질적인 병렬 처리 시스템까지의 아주 다양한 형태를 가지고 있다. 특히 최근에는 공개된 운영체제 코드인 리눅스를 바탕으로 한 클러스터 개발이 가속화되고 있다. 이에 따라서 리눅스 클러스터 환경에서 병렬 처리 및 분산 처리를 지원하기 위한 연구 또한 활발하게 진행되고 있다.

클러스터 환경에서의 병렬 및 분산처리는 다음과 같은 여러 가지 중요한 장점들을 가지고 있다:

- 클러스터 내의 프로세서 노드들은 서로 다른 응용에 광범위하게 사용될 수 있는 완전한 시스템일 수 있다.
- 클러스터 컴퓨팅은 매우 큰 시스템으로도 확장이 가능하다. 현재까지는 리눅스를 사용하면서 4개 이상의 프로세서 노드로 된 SMP(symmetric multiprocessor)를 찾는 것은 어렵지만 16개 이상의 노드로 클러스터를 구성할 수 있게 해주는 네트웍 하드웨어는 쉽게 찾을 수 있다. 앞으로는 간단한 작업으로 수 백 개에서 수 천 개의 노드를 네트웍으로 묶는 것이 가능할 것이다. 사실 전체적인 인터넷을 사실상의 거대한 클러스터로 볼 수 있게 될 것이다.
- 클러스터 내에서 문제가 되는 프로세서 노드를 교체하거나 제거하는 것은 기존의 SMP에서 노드를 교체하거나 제거하는 것에 비해서 훨씬 간단하다. 이것이 클러스터가 제공할 수 있는 높은 가용성을 보여 주는 한 예이다.

그렇다면 클러스터가 가지는 많은 장점에도 불구하고 클러스터가 보편화되지 못하고 있는 이유는 무엇인가? 그것은 클러스터가 다음과 같은 문제점도 동시에 가지고 있기 때문이다.

- 일부를 제외하고는 네트웍 하드웨어들이 병렬 처리를 위해서 설계되어 있지 않다. SMP와 비교할 때 latency가 매우 높고 대역폭이 비교적 낮다. SMP의 latency는 수 마이크로 초를 넘지 않지만 클러스터에서는 수백에서 수천 마이크로 초까지 된다. 또한 대역폭에 있어서도 SMP는 초당 100M 바이트 이상이지만 가장 빠른 ATM 네트웍조차도 5배 이상 느리다. 하지만 FC(Fibre Channel)가 표준화되고 관련 장치들이 보편화되면 이 문제는 해결이 될 것으로 보인다.
- 단일 시스템 이미지 제공을 위한 소프트웨어 지원이 아직 미미한 수준이다.

클러스터는 아주 큰 잠재력을 가지고 있지만, 현실적으로 대부분의 응용에서 그 잠재력을 제대로 활용하는 것이 매우 어렵다. 물론 일부 응용에서는 클러스터를 통해서 상당한 성능 향상을 이루고 있는 것 또한 사실이다.

3. 리눅스 파일 시스템

리눅스 클러스터 환경에서 병렬 분산 파일 시스템에 대해서 알아보기 전에 리눅스에서 사용하고 있는 파일 시스템을 먼저 알아보기로 한다.

리눅스 파일 시스템의 가장 중요한 특징 중의 하나는 Virtual File System(VFS)을 인터페이스 층으로 하여 커널에 단일화 된 파일 시스템 인터페이스를 제공하므로써 ext, ext2, xia, minix, umsdos, msdos, vfat, proc, smb, ncp, iso9660, sysv, hpfs, affs, ufs 등의 다양한 파일 시스템을 지원한다는

것이다. 리눅스에서 각각의 서로 다른 파일 시스템들은 단일 엔티티로 보여지는 하나의 계층적 트리 구조로 구성되며, 리눅스는 파일 시스템이 mount 될 때마다 새로운 파일 시스템을 파일 시스템 트리에 추가하게 된다. 모든 파일 시스템은 하나의 디렉토리에 마운트 되며, 마운트 된 파일 시스템의 파일들은 그 디렉토리의 내용을 구성한다.

리눅스는 Minix 운영체제를 토대로 개발되었기 때문에 리눅스에서 지원한 최초의 파일 시스템은 Minix 파일 시스템(Minix file system)으로 최대 파일 크기를 64M바이트까지 지원했으며, 파일 이름은 14자까지로 제한되어 있었다. 이 파일 시스템은 속도가 느린 단점을 가지고 있다.

1992년에 Minix의 여러 가지 단점을 보완하여 리눅스의 최초 파일 시스템인 Extended File System (EXT)이 소개되었다. 이 파일 시스템은 최대 파티션 크기와 최대 파일 크기를 2G Byte까지 지원하고 파일 이름도 255자까지 허용하였으나, Minix 파일 시스템보다 더 느린 속도와 파일 시스템의 심각한 단편화 현상(fragmentation)까지 발생하여 사용에 많은 문제점을 보였다.

1993년에 Remy Card가 EXT의 문제점을 보완하여 개발한 것이 EXT2 파일 시스템(The Second Extended File System)으로 현재까지 가장 널리 쓰이고 있다. 이 파일 시스템은 파티션을 여러 개의 블록 그룹으로 나누고 각 블록 그룹은 슈퍼블록의 복사본과 inode, 그리고 테이블 블록들을 가진다. 각 블록 그룹은 데이터 블록들을 그들의 inode에 가깝게 유지시키고, 파일의 inode를 그들의 디렉토리에 가깝게 위치시키도록 하는 정보를 유지하여, 파일을 위치시키는 시간을 최소로 줄이고 데이터 접근 속도를 빠르게 해준다.

다음 장에 소개될 대부분의 리눅스 클러스터 환경의 분산 병렬 파일 시스템은 리눅스의 VFS에 접속시켰거나 EXT2 위에서 운영되도록 개발되었다.

4. 분산 병렬 파일 시스템

4.1 개요

리눅스 클러스터 환경에서의 병렬 분산 파일 시스템은 대부분이 기존의 다양한 UNIX 클러스터 환경에서 개발된 파일 시스템들로서 이들을 리눅스 환경으로 이식 한 것들이 대부분이다. 물론 최근에는 리눅스 환경에 직접 개발된 것들도 있다.

리눅스 운영체제 환경에서 개발된 병렬 파일 시스템으로는 PVFS(Parallel Virtual File System), FUFS(Fairly Usable File System), CrownFS가 있다. 그리고 Unix 환경에서 개발된 것을 리눅스환경에 이식한 분산 병렬 파일 시스템으로는 ParFiSys(Parallel File System), PIOUS(Parallel Input/Output System), 그리고 Coda 파일 시스템이 있다.

Clemson 대학교에서 개발한 PVFS는 리눅스 클러스터 환경에서 사용하는 병렬 파일 시스템으로 커널 수준이 아닌 사용자 수준에서 구현되어 있다. 이 파일 시스템은 클러스터 전체에 이름 공간의 일관성을 제공하고, 사용자가 데이터의 스트라이핑을 조절할 수 있는 기능도 제공한다[12, 19].

FUFS는 Purdue 대학에서 개발한 리눅스용 병렬 파일 입출력 시스템으로 VFS 아래의 커널 모듈에 통합시키므로써 사용자 수준의 병렬 파일 시스템에 비해 시스템 오브헤드를 크게 감소시켰다[17].

ParFiSys는 스페인의 UPM(Universidad Politechica de Madrid)에서 개발된 병렬 파일 시스템이다. 이 시스템은 병렬 분산된 입출력 하드웨어를 최대로 활용하기 위해서 개발되었으며 TCP/IP를 기반으로 프로세스간의 통신 부분을 구현하였다. 몇 대의 SUN 워크스테이션이 입출력 노드 혹은 처리기 노드로 동작하면서 하나의 논리적인 병렬 파일 시스템을 형성하게 되어 있다. 그리고 현재 리눅스 클러스터 환경으로 이식 중에 있다.

Emory 대학에서 개발된 PIOUS는 PVM(Parallel Virtual Machine)을 기반으로 운용되는 병렬 파일

시스템이다. 이 시스템은 비동기 적인 작업 수행 모델을 채택하여 성능을 향상시켰다[3,6,8,9,18].

Coda는 카네기 멜론 대학에서 개발한 분산 파일 시스템으로 AFS (Andrew File System)를 기반으로 하고, 초기에 Mach 커널 위에서 개발되었으나 현재 리눅스와 FreeBSD 등에 이식되었다[14].

ETRI에서 연구 개발한 CrownFS는 VOD 서버를 위해 개발된 병렬 파일 시스템이다. 이 시스템은 Myrinet이라는 고속 스위칭 네트워크를 통해서 서버와 클라이언트 노드들을 연결하고 있다. 이 파일 시스템은 VOD 서비스용에 맞게 개발되어 다른 응용에 적용하기는 쉽지 않은 구조를 가지고 있다[13].

이 이외에도 여러 가지 분산 병렬 파일 시스템들이 연구 개발되었고 지금 많은 연구가 진행되고 있다. 리눅스 환경에서 구현은 되지 않았지만 분산 병렬 파일 시스템 연구 중 중요한 것들로는 HiDIOS(High Performance Distributed Input Output System)[11], Zebra, GPFS(Galley Parallel File System) [15], xFS 등이 있다.

HiDIOS 파일 시스템은 Fujitsu AP1000 다중 컴퓨터를 위해 만들어진 병렬 파일 시스템이다. 이는 수퍼 컴퓨터에서의 작업병목인 디스크로부터의 입출력을 해결하기 위해서 만들어졌다[11].

Zebra는 버클리 대학에서 개발한 스트라이프 네트웍 파일 시스템이다. 이 시스템은 LFS (Log-structured File System)의 개념을 적용하여 작은 파일들에 대해서도 서버의 효율을 높임으로써 저장 서브시스템의 쓰기 성능을 향상시켰다. 그리고 소프트웨어 RAID (Redundant Array of Inexpensive Disks)의 스트라이핑과 패리티 개념을 적용하여 비교적 낮은 성능의 디스크 배열로부터 높은 성능과 가용성을 얻었다.

GPFS는 Dartmouth 대학교에서 개발된 병렬 파일 시스템으로 특히 작업 부하에 대한 연구 결과를 바탕으로 과학 연산을 위한 응용 프로그램에 적합하도록 구현되었다. GPFS는 병렬 수퍼 컴퓨터에서 수행될 수 있도록 설계되었으며 현재 IBM RS/6000s 망을 이용한 IBM SP-2 병렬 수퍼 컴퓨터에서 수행되고 있다[15].

xFS는 WAN 환경에서 계층 구조로 이루어진 초 대용량의 네트워크 파일시스템으로 버클리 대학에서 개발되었다. 기존의 네트워크 파일시스템은 단층 구조로 되어 있어서 이를 WAN과 대용량의 저장장치에 적용하기에는 파일 시스템의 프로토콜이 좋지 못하고, 중앙 서버에 너무 의존적이었다. xFS는 호스트 컴퓨터들이 계층 구조로 이루어져 있고 지리적 근접도를 기반으로 한 클러스터링을 사용하여 지역 접근과 원격 접근을 구분하였고 확장성을 높였다. 그리고 캐쉬의 일관성을 위한 프로토콜을 제공하는데, WAN을 통한 데이터 전송을 최소화하기 위해 멀티프로세서 스타일의 캐쉬 일관성 기법을 사용했다[21].

다음 절에서는 대표적인 분산 병렬 파일 시스템들의 기능과 내부 구조에 대해서 구체적으로 살펴보기로 한다.

4.2 PVFS(Parallel Virtual File System)

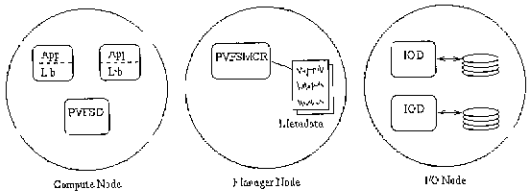
PVFS(Parallel Virtual File System)는 미국의 Clemson 대학의 병렬 구조 연구실에서, 리눅스 클러스터 시스템인 Beowulf를 위한 파일 시스템으로 설계되었다. PVFS는 다음과 같은 특징을 가진다.

- 기존의 응용프로그램에 대한 바이너리 수준의 호환성 제공
- 스트라이핑에 대한 사용자 설정 기능 제공
- 다양한 사용자 인터페이스 제공

PVFS는 사용자 수준에서 구현되었다. 이로 인해서 커널을 수정하지 않고도 설치 및 사용이 가능하다. 또한 데이터의 전송은 TCP/IP 프로토콜을 사용하므로, 서로 다른 메시지 전달 방법을 가진

시스템에서도 TCP/IP를 지원을 한다면 쉽게 이식이 가능하다.

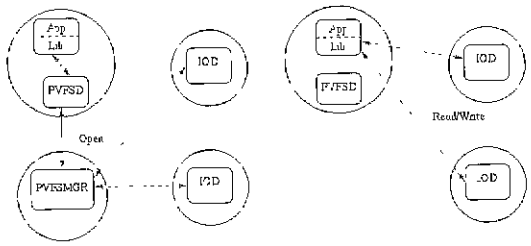
PVFS는 크게 관리자 노드(Manager node), 클라이언트인 처리 노드(Compute node), 그리고 서버인 입출력 노드(I/O node, Server)의 세 가지 부분으로 구성되어 있다 (그림 1).



(그림 1) PVFS의 세가지 구성 요소

파일은 입출력 노드에 병렬 분산 저장되어 있고, 처리 노드에서 이 파일을 액세스해서 처리하는 구조로 되어 있다. 전체 파일 시스템의 메타데이터는 관리자 노드에서 관리하게 되는데, 관리자 노드의 부하가 크지 않을 경우 처리 노드 또는 입출력 노드 중의 하나가 이를 겸하게 할 수도 있다.

PVFS 파일을 처리 노드에서 액세스하려면, 먼저 관리자 노드에서 해당 파일의 메타데이터를 가져온다. 메타데이터에는 파일의 접근 권한, 소유자, 파일의 스트라이핑 상태 등의 정보가 들어 있다. 일단 메타데이터를 가져오면, 실제 파일의 내용을 전달할 때는 관리자 노드는 개입하지 않고, 처리 노드와 입출력 노드 사이에 직접 데이터를 전달하게 된다 (그림 2).



(그림 2) PVFS에서의 파일 액세스

PVFS를 사용할 때는 시스템의 LD_PRELOAD 환경 변수에 PVFS 라이브러리를 로드해서 시스템 호출 랩퍼(system call wrapper)를 사용한다. 시스템 호출 랩퍼가 로드되면 모든 사용자 프로그램의 시스템 호출은 시스템 호출 랩퍼를 통과하게 되는데, 이 중에서 PVFS에 관계되는 시스템 호출들은 PVFS 라이브러리에서 처리하고, 일반적인 시스템 호출들은 시스템으로 전달되어 처리하게 된다.

시스템 호출 랩퍼를 사용하므로써 얻는 이점은 커널의 수정없이, 또한 응용프로그램의 수정없이 PVFS를 사용할 수 있다는 것이다.

그리고 PVFS는 CHARISMA(CHARacterize I/O in Scientific Multiprocessor Applications) Project[4]에서 얻어낸 결과들 중에서 병렬 처리 작업의 파일 액세스 특성 중에서 'simple strided pattern'을 참고하여 응용프로그램에서 파일의 스트라이핑을 조절할 수 있는 인터페이스를 제공하고 있다. 'simple strided pattern'이란 병렬 처리 프로세스의 파일 액세스 특성이 파일 내부에서 일정한 크기(64-256 bytes 정도)의 조각을 일정한 간격을 두고 액세스 하는 것을 말한다.

PVFS는 파일의 병렬 입출력을 위해서 다양한 기능을 제공하지만 파일 전송을 위해서 TCP/IP를 사용하므로 해서 오버헤드가 크고 데이터 캐싱에 대한 고려가 없어서 성능 개선의 여지가 많이 남아 있다.

4.3. FUFS (Fairly Usable File System)

FUFS(Fairly Usable File System)는 Purdue 대학에서 개발된 리눅스용 병렬 파일 입출력 시스템이다. PUFS는 리눅스 시스템 환경에서 MPI-IO와 같은 병렬 라이브러리를 제공하기 위한 완전한 기능을 갖춘 병렬 파일 시스템이다. FUFS는 사용자 수준 병렬 라이브러리의 시스템 콜 오버헤드를 줄이기 위해서 VFS 아래의 커널 모듈로 만들

어져 있다.

FUFS는 빠른 동기화와 전체적인 통신을 제공하는 AFN(Aggregate Function Networks)을 근간으로 구성되어 있다. AFN은 노드들 간의 통신을 매우 효율적으로 제공하는 특징을 가지고 있다. 그리고 FUFS는 빠른 동기화와 글로발 통신을 제공하는 PAPERS(Purdue's Adapter for Parallel Execution and Rapid Synchronization)을 사용한다. PAPERS는 통신 지연이 약 3 μsec 정도이고, broadcasts, reductions, scans 등과 같은 효율적인 collective 네트웍 기능도 제공한다. 하지만 다소 낮은 대역폭(1.6Mb/s)을 가지고 있어서 매우 큰 데이터 전송에는 기존의 표준 이더넷을 사용한다.

FUFS의 노드는 자율적 처리기와 함께 각자의 디스크를 가지고 있다. FUFS에서의 파일들은 각각의 독립적인 세그먼트들로 나누어져 있고 각 세그먼트는 정확하게 한 노드에 존재하며, 파일 및 노드마다 정확히 하나의 세그먼트가 있다. 정규적인 FUFS 파일로부터 생성된 파일 데이터는 클러스터 노드들에 나누어 저장된다. 모든 메타데이터와 클러스터링 되지 않은 정보들은 분산되고, 복제된 inode 들을 통해 일관성을 유지한다.

FUFS는 다음과 같은 특징이 있다.

- 통합된 이름 공간(integrated namespace)을 지원
  FUFS는 세그먼트 트리를 이용하여, 통합된 하나의 계층적인 이름 공간를 유지한다. FUFS에서는 어떠한 마운트 지점이라도 클러스터 노드로부터 마운트 될 수도 혹은 언마운트 될 수도 있다.
- 유지보수 유틸리티(Maintenance utilities) 제공
  cp, mv, mkdir, 등과 같은 모든 표준 리눅스 유틸리티들도 FUFS에서 잘 작동되며, 스트라이프된 FUFS 파일들에서도 정상적으로 동작한다.
- Multi-API를 지원
  FUFS는 inode 터널링을 이용하여 많은 API들을 직접 지원할 수 있게 되어 있다. 뿐만 아니라, 효율적으로 병렬 파일 입출력 라이브러리를 구현할 수 있게 일반적인 병렬 인터페이스를 개발하였다. 많은 경우에, 이러한 병렬 라이브러리 루틴은 매우 적은 오버헤드를 가진 간단한 랩퍼 함수들로서 만들어 질 수 있다.

FUFS는 리눅스 2.0.3.이 동작중인 PC 8대를 묶은 클러스터 상에서 설계 및 구현되었고, 각 노드들은 AFN과 이더넷을 통해 연결되어 있다. AFAPI 런타임 라이브러리를 통한 저수준의 루틴을 가진 AFN을 이용하여 내부적인 통신을 하며, 데이터의 전달은 부분적으로 AFN을 통해 이루어지고, 또한 TCP/IP 기반의 이더넷을 이용하기도 한다.

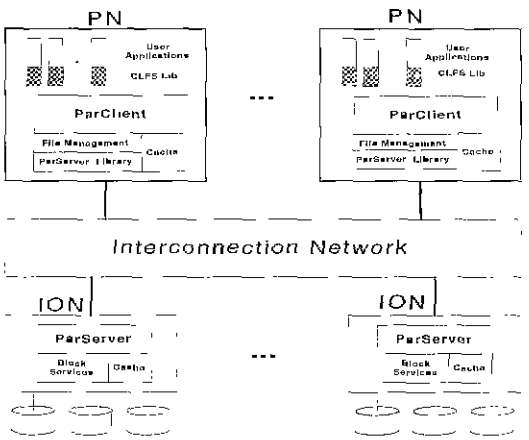
4.4 ParFiSys (Parallel File System)

ParFiSys는 CEE ESPRIT 프로그램의 일환으로 스페인의 UPM(Universidad Politechica de Madrid)에서 개발된 병렬 파일 시스템이다. 이 시스템은 높은 대역폭을 요구하는 과학 계산 분야의 입출력 서비스를 처리해주기 위해서 개발된 것으로, 처리기 노드와 입출력 노드 사이에 메시지 전달 방식을 이용하는 다중 컴퓨터 시스템에서 병렬성을 줄 수 있도록 되어있다. 그리고 POSIX 표준에 따라 만들어졌음으로 이식성이 뛰어나다. 현재 이 병렬 파일 시스템은 소결합의 분산 시스템과 밀결합 분산 메모리 시스템 그리고 밀결합 공유 메모리 시스템에 구현되어 있다.

ParFiSys는 TCP/IP를 기반으로 프로세스간의 통신 부분이 구현되었으며, 몇 대의 Solaris가 탑재된 SUN 워크스테이션이 입출력 노드 혹은 처리기 노드로 동작하면서 하나의 논리적인 병렬 파일 시스템을 형성한다. 표준 POSIX 환경에서 개발된 병렬 파일 시스템으로 리눅스 클러스터 환

경으로 이식이 가능하며 POSIX의 다중 쓰레드로 구현하여 병렬성을 높였다.

ParFiSys는 크게 프로세스 노드에서 파일 서비스를 담당하는 ParClient 부분과 입출력 노드에서 자료 블록의 입출력 서비스를 담당하는 ParServer 부분, ParServer에 대해 디스크 장치에 대한 투명성을 보장해주는 CDS(Concurrent Disk System)으로 나뉘어진다(그림 3).



(그림 3) ParFiSys의 구조

사용자 응용프로그램은 ParFiSys run-time 라이브러리를 호출하므로써 사용자와 ParClient가 자료를 주고 받을 수 있게 되어 있다. 그리고 ParClient는 ParServer 라이브러리를 호출하므로써 서로 자료를 주고 받을 수 있게 되어 있다.

ParFiSys는 고성능 파일 입출력을 위해서 다음과 같은 파일 입출력 기능을 제공한다.

- 세그먼트 파일
  표준 파일 포인터 크기(32 비트)로 접근할 수 없는 파일에 대한 접근을 지원한다.
- 다중 파일
  특수한 파일 종류로 여러 개의 서브파일의 집합으로 된 특수 파일에 병렬로 접근이 가능하게 되어 있다. 파일의 안전한 병렬 접근을 위해서 여러 프로세스가 동시에 겹치지 않고 접근이 가능하도록 제어한다.
- 글로벌 파일
  MPP나 클러스터 시스템에서 Unix와 같이 파일 포인터의 공유를 가능하게 한다.

ParFiSys에서는 적은 양의 읽기에 대한 성능 향상을 위해 ParClient에서 선반입 기법을 이용하며, 많은 양의 쓰기에 대한 성능 향상을 위해 Write-before-full 기법을 이용한다.

그러나 사용자 수준에서 설계, 구현되므로써 병렬성에 반해 성능이 낮아 개선의 여지가 많이 남아있다.

4.5 PIOUS(Parallel Input/Output System)

PIOUS는 UNIX/NT 클러스터링을 위한 소프트웨어 패키지인 PVM(Parallel Virtual Machine)을 위해 미국 Emory 대학에서 개발된 병렬 입출력 시스템이다. PVM은 비대칭적인 UNIX/NT 시스템을 네트워크로 묶어서 하나의 고속 병렬 처리 컴퓨터로 동작하게 하기 위해 만든 소프트웨어 패키지이다[10]. PIOUS는 PVM 위의 병렬 파일 시스템으로 구현되었다. PVM 응용프로그램은 병렬 파일 시스템에 대해 투명하게 접근할 수 있다.

PIOUS는 다음과 같은 기능을 제공하고 있다.

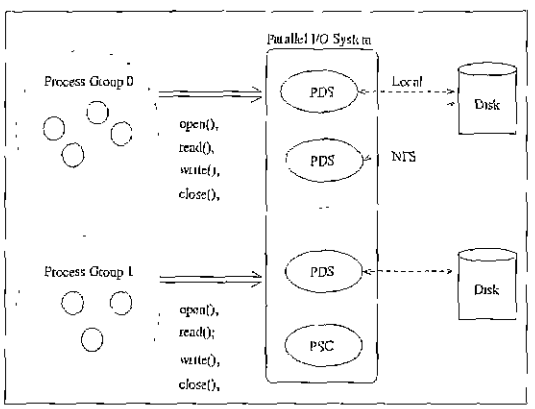
- 2 차원 파일 오브젝트와 응용프로그램에 대한 논리적 파일 인터페이스 제공
- 파일에 동시 접근시 일관성 유지 기능 지원
- 트랜잭션 기능 제공
- 분산된 파일을 관리하기 위한 향상된 파일 관리 기능 제공

PIOUS는 여러 하드웨어 플랫폼과 병렬 처리 환경에 대해 쉽게 포팅 할 수 있도록, 하드웨어와

소프트웨어의 세부적인 부분에 대한 의존도를 최소로 하여 같은 Unix 계열인 리눅스 환경에도 이식이 가능할 것이다. 특히 데이터의 전송과 메시지 전달 부분에 대해서는 PIOUS가 운용되는 시스템의 하드웨어와 운영체제가 충분히 그 역할을 담당할 수 있다고 가정하고 있기 때문에 새로운 프로토콜 사용 등에 따른 이식하기 어려운 부분은 최대한 배제하고 있다.

고성능을 얻기 위해 PIOUS는 비동기적인 작업 수행 모델을 채택하고 있다. PIOUS의 각 부분들은 서로에 대해 독립적으로 동작하며 동기화 작업에 의해 생기는 오버헤드를 줄이고 있다. PIOUS의 독립적인 각 부분들의 병렬 동작성은 각 프로세스가 파일에 독립적으로 접근하는 MIMD(Multiple Instruction Multiple Data) 스타일의 파일 시스템 인터페이스를 가능하게 한다.

PIOUS 에서도 파일 스트라이핑을 지원하고 있다. 하나의 파일을 여러 대의 병렬 시스템에 분산 저장하므로써 저렴한 가격의 일반적인 저장 매체를 사용하여 고성능의 RAID 시스템에서와 같은 데이터 전송 속도의 향상을 보여주고 있다.



(그림 4) PIOUS의 소프트웨어 구조

PIOUS의 구성요소는 서비스 조종자, 데이터 서버, 그리고 클라이언트 프로세스와 링크된 라이브러리 루틴들이다(그림 4). 각 구성요소들 사이의 메시지 전달은 PIOUS가 운용되는 시스템의 운영체제와 하드웨어에서 제공한다고 가정하며, PIOUS의 데이터 서버는 시스템 고유의 파일시스템을 사용해서 디스크에 파일을 저장하고 접근하게 된다. 즉 PIOUS는 시스템의 세부적인 부분에 대해서는 해당 시스템의 운영체제와 하드웨어에 의존하고 있다.

프로세스가 파일을 열기 위해 라이브러리 함수 open()을 실행하면, 파일의 메타데이터와 파일에 접근하기 위해 필요한 정보를 얻기 위해 PSC(PIOUS Service Coordinator)에 접근 한다. 일반적으로 PSC는 전체 클러스터 시스템에서 하나만 존재하지만, 만약 부하가 많이 걸릴 경우에는 PSC를 복제해서 여러 개를 둘 수도 있다. 이때 파일의 이름을 인자로 hash 함수를 사용해서 각 PSC에 부하를 분산한다.

PDS(PIOUS Data Server)는 파일이 저장되어 있는 각 시스템마다 존재하며, 로컬 파일에 대해 트랜잭션에 바탕을 둔 접근을 제공한다. PDS는 독립적으로 동작하며, 시스템의 병렬성과 확장성을 위해 다른 PDS와 통신하지는 않는다.

PIOUS를 사용하는 클라이언트 프로세스는 PIOUS 라이브러리와 링크 되어 있다. 클라이언트 프로세스가 PIOUS 파일에 액세스하면 라이브러리는 이를 PSC/PDS 서비스 요청으로 변환한다.

PIOUS는 제한된 수의 노드로 구성된 시스템에서 구현하고 성능평가를 했으므로 확장성에 대한 확실성을 보여주지 못하고 있다.

4.6 CrownFS

리눅스 클러스터용 병렬 파일 시스템 중에서 국내에서 개발된 것으로 ETRI에서 연구 개발한 CrownFS이 있다. 이 시스템은 Crown이라고 하는 VOD 서버를 위해 개발된 병렬 파일 시스템이다.

CrownFS는 M-worker와 S-worker 및 클라이언트

로 구성되어 있다. M-worker는 클라이언트와 저장 서버 노드 사이의 스트림을 조정하는 역할과 각 클라이언트의 요구를 받는 노드이다. 즉 클라이언트가 요구한 스트림을 저장 서버에서 받아, 전달하는 역할을 한다. 저장 서버인 S-worker는 MPEG-1 파일을 디스크에 분산 저장하는 역할을 담당한다.

구현된 Crown 시스템은 9대의 고속 PC로 이루어지며 1대의 M-worker (Master worker)와 8대의 S-worker(Slave worker)로 구성되어 있다. 네트워크는 Myrinet이라는 고속 스위칭 네트워크 장비를 사용해서 연결된다. 비디오데이터는 8대의 S-worker의 디스크 상에 분산 저장되며 클라이언트가 요구한 데이터를 읽어서 M-worker에게 전달하고 M-worker는 이를 버퍼에서 클라이언트에 전달하기 위해 순서대로 정리한 다음 클라이언트에게 보낸다.

CrownFS이 탑재된 시스템은 VOD 서비스를 위해서 개발되었지만 아직 VCR 동작 수준의 성능을 보여주지는 못하고 있다.

4.7 Coda Distributed File system

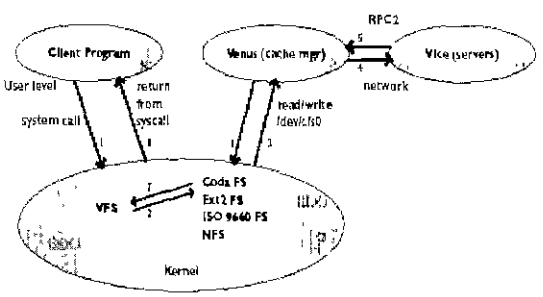
Coda는 카네기 멜론 대학의 M. Satyanarayanan 팀에 의해 개발되고 있는 분산 파일 시스템이다. 이 분산 파일 시스템은 Unix 환경에서 개발된 것으로, 같은 Unix 계열인 리눅스 환경에도 이식되었다.

Coda의 주된 응용 분야에는 FTP 미러 사이트, WWW 복제 서버와 네트웍 컴퓨터 등에 활용하는 것이다. 여러 클라이언트가 사용하는 FTP 사이트가 미러 사이트를 가지고 있다면 기존에는 일일이 업데이트된 파일을 각 미러 사이트에 전송해야 했지만, FTP 미러 사이트에서 Coda 파일 시스템을 사용하면 가장 최신의 파일로 자동으로 업데이트한다. WWW 복제 서버에서의 Coda 파일 시스템은 사용자가 많은 WWW 서버의 부하를 여러 WWW 서버로 분산시키기 위해 여러 대의 서버를 사용할 때, 한 WWW 서버의 내용을 업데이트하면 자동으로 다른 서버의 내용도 업데이트된다. 네트웍 컴퓨터에서의 Coda 파일 시스템의 캐시정책은 네트워크 부하량을 줄여서 전체적인 속도를 향상시킬 수 있게 한다.

유닉스 환경에서의 대표적인 분산 파일 시스템인 NFS는 개개의 서버가 파일시스템을 export하고 클라이언트가 그것을 마운트해서 사용하게 되는데 반해서, Coda는 개개의 서버를 의식하지 않고, Coda 파일시스템을 클라이언트에서 접근하면 전체 Coda 파일시스템 서버를 하나로 보이게 한다. 클라이언트의 입장에서 Coda 파일은 /coda 디렉토리에 저장되고, 각 파일이 어떤 특정 서버에 있는지에 대한 정보없이 각 파일을 접근할 수 있다.

이렇게 서버마다 각각 하나의 디렉토리를 마운트하지 않고 전체 분산 파일 시스템을 하나의 디렉토리로 마운트하는 방식은 Coda의 선배격인 AFS(Andrew File System)에서 시작되었다. 하나의 디렉토리에 전체 분산 파일 시스템을 마운트하게 되면 각 클라이언트가 보는 파일 이름이 모두 동일하게 된다. 따라서 모든 사용자가 보는 파일 트리가 같은 구조를 가지게 되는 것이다[5].

그림 5.는 Coda 파일에 대한 액세스를 어떻게 처리하는지에 대한 설명이다.



(그림 5) Coda의 파일 액세스 과정

클라이언트 프로그램이 파일을 열기 위해 커널에 시스템 호출(system call)을 요청하면 커널의

VFS(Virtual File System)는 해당 디렉토리의 파일 시스템을 호출하게 된다. 클라이언트 프로그램이 요청한 파일이 Coda 파일이면 Coda FS에 대해 요청을 하게되고 Coda의 캐시 관리자인 Venus가 캐시를 확인하고 해당 파일이 캐시 내에 존재한다면 바로 알리고, 캐시 내에 없다면 해당 서버의 Vice에 신호를 보내서 파일을 읽어 오게 된다.

Coda는 또한 서버의 오류나 네트워크 단절로 인한 에러에 대해 유연하게 대처한다. 처음 파일을 요청하면 파일의 내용과 속성 등의, 파일에 액세스하는데 필요한 모든 정보가 캐시에 저장된다. 이 후에 그 파일에 대한 접근은 모두 캐시된 파일에 행해진다. 파일이 변경되고 파일을 닫았다면 Venus는 변경된 파일을 서버로 전송해서 업데이트하게 된다. 하지만 네트워크나 서버의 오류로 전송이 불가능하다면 사용자에게 에러메시지를 보이지 않고, CML(Client Modification Log)에 기록해둔다. 이후에 네트워크 연결이 이루어지면 자동으로 서버의 파일을 업데이트한다.

조사결과에 따르면, 대부분의 파일 액세스는 읽기 전용이고, 파일을 수정하는 등의 쓰기 작업은 상대적으로 적은 부분을 차지하고 있다. 따라서, Coda의 이러한 캐시정책은 대부분의 작업에서 매우 효율적이다.

최초에 이러한 캐시정책은 CMU에 설치된 AFS가 네트워크망의 노화로 인해 자주 네트워크 연결 오류를 일으키는 것에 대한 해결책으로 고안된 것이었다. 그 후 이러한 방법이 모빌 클라이언트 환경에도 적용 가능하다는 사실을 알아내었다.

서버에 설치된 Coda의 물리적인 파일 시스템은 일반적인 UNIX 파일 시스템과는 다른 Coda만의 독자적인 파일 시스템을 사용하고 있다. 서버상의 Coda 파티션 내에서는 일반적으로 10MB 안팎의 크기를 가지는 볼륨 단위로 파일과 디렉토리를 관리한다. 볼륨은 고유한 이름과 ID를 가지고 있고, 자신만의 루트 디렉토리와 그에 따르는 하위 디렉토리 및 파일을 가지고 있다. 볼륨은 클라이언트의 Coda 파일시스템 상의 어떤 곳에도(/coda 디렉토리의 아래라면) 마운트 할 수가 있다. 하지만 기존에 존재하고 있는 디렉토리로는 마운트 되지 않는다.

이러한 개념은 Macintosh나 Windows의 "Network drive and volume"과 흡사하나, 클라이언트에게는 마운트 포인트가 보이지 않는다는 점에서 다르다. 클라이언트에게는 /coda 디렉토리의 일반적인 파일처럼 보이게 된다.

Coda는 세 개의 32비트 정수로 된 Fid로 각 파일을 구분한다. Fid는 VolumeId, VnodeId, Uniquifier로 이루어진다. VolumeId는 파일이 위치하고 있는 볼륨을 나타내고, VnodeId는 파일의 Inode 번호를 나타내며, Uniquifier는 VSG(Volume Storage Group)들 사이에서 같은 파일이 복사되어 있을 경우에 어떤 파일이 가장 최근의 파일인지 확인할 때 쓰인다. 이러한 Fid는 전체 Coda 서버 클러스터에서 유일하다.

VSG(Volume Storage Group)는 요청빈도가 높은 파일에 대해서 여러 대의 Coda 파일 서버에 같은 파일을 복사해 두므로써 클라이언트가 그 파일에 액세스할 때 가용성을 높인다. 또한 한 서버가 다운되더라도 다른 서버가 계속 그 파일을 서비스할 수 있으므로 클라이언트가 시스템상의 변화를 전혀 의식하지 못하고 계속 작업을 수행할 수 있도록 해준다. 또한 여러 개의 복사본을 가지는 파일에 대해 업데이트가 일어나면 자동적으로 전체 서버의 해당 파일을 업데이트한다.

4.7 HiDIOS(High Performance Distributed Input Output System) 파일시스템

HiDIOS(High Performance Distributed Input Output System) 파일 시스템은 오스트레일리아 국립대학의 PIOUS 프로젝트에서 나온 결과이다.

HiDIOS 파일 시스템은 Fujitsu AP1000 다중컴

퓨터를 위해 만들어진 병렬 파일 시스템이다. 수퍼 컴퓨터에서 작업의 병목은 디스크로부터의 입출력이기 때문에 이에 관한 많은 병렬 파일 시스템들이 고안되어졌다. HiDIOS 파일 시스템은 그 중의 하나이다.

기존의 병렬 파일 시스템은 스트라이핑을 통한 파일 저장의 병렬화를 통해서 전송 대역폭을 확장하는 방법이 널리 사용되었다. 하지만 이 경우에 파일의 입출력 시에 파일 시스템 관리 서버와의 통신을 피할 수 있는 방법이 거의 없어서 입출력이 과다할 경우 파일 시스템 관리 서버와의 통신에서 병목이 발생할 수밖에 없다. HiDIOS 파일 시스템은 옵션 보드라는 특별한 하드웨어를 통해서 여러 디스크가 하나의 거대한 디스크로 보이도록 하므로써 파일 시스템의 설계가 간편해지고, 견실성에서 스트라이핑을 사용한 파일시스템에 비해 앞선다.

물론 이상적인 경우, 파일 수준의 병렬화가 더 높은 전송률을 나타내겠지만, 파일 시스템을 유지하는데 프로세서에 많은 부하가 걸리고, 파일 시스템 관리 서버와의 통신에 의한 병목현상 때문에 실제 작동 시에는 오히려 전송률이 저하되게 된다.

HiDIOS 파일 시스템에서는 메타데이터를 사용하여 파일 시스템 관리 서버와의 통신 회수를 최소화하므로써 병목 현상을 방지한다. 현재 열어서 사용하고 있는 파일에 대해서 메타데이터를 메모리 상에 저장하여 파일을 읽거나 쓸 때 메타데이터를 참조하므로써 파일 시스템 관리 서버와의 통신을 줄인다. HiDIOS 파일 시스템에서 파일 시스템 관리 서버와의 통신은 현재 열어서 사용 중인 파일이 변경되더라도 변경 부분이 현재 사용되고 있는 부분에 영향을 미치지 않는다면 업데이트를 하지 않는다

예를 들면 파일이 기존의 부분에는 변경이 없고, 다른 프로세서에서 추가하여 길이가 길어졌을 경우에는 업데이트를 하지 않고 메타데이터에 기록된 파일의 끝까지 읽고, 그 이후를 읽으려 할 때만 파일 시스템 관리 서버와 통신을 해서 메타데이터를 업데이트하게 된다. 캐쉬된 메타데이터를 업데이트하는 것은 파일의 길이가 짧아졌을 경우에만 일어난다. 메타데이터를 사용함에 있어서도 아주 간단한 구조를 사용하므로써 메모리 사용량과 프로세서의 부하를 줄일 수 있다.

HiDIOS에서 지원해 주는 유틸리티를 살펴보면 apsh라는 간단한 shell을 제공하며 tlist, trename, tmkdir, trmdir등과 같은 간단하면서도 편리한 유틸리티 및 tbuffer, trepack과 같은 버퍼링의 속도와 관련된 것도 몇 개를 지원한다. 사용자는 일반적으로 40Mb/sec정도의 성능을 기대할 수 있으며 대규모의 입출력 작업을 한다면 이 이상의 성능도 기대할 수 있다.

5. 결 론

본 고에서는 리눅스 클러스터와 함께 연구 개발이 활발하게 진행되고 있는 분산 병렬 파일 시스템에 대해서 살펴보았다. 리눅스의 활용이 아직 본격적인 궤도에 오르지 않은 상황이라, 리눅스 환경만을 고려해서 개발된 분산 병렬 파일 시스템은 그렇게 많지 않지만 기존의 Unix환경에서 개발된 많은 분산 병렬 파일 시스템들이 리눅스 환경으로 이식되고 있다. 물론 일부 분산 병렬 파일 시스템들은 처음부터 리눅스 클러스트 환경에서 연구 개발되기도 했지만 아직 학교나 연구소 수준에 머물러 있는 형편이다.

국내에서도 리눅스를 이용한 웹 서버 구축은 많이 이루어지고 있으나 대부분 소규모의 서버에만 활용이 됨으로 해서 분산 병렬 파일 시스템의 요구가 미약한 편이다. 하지만 국내에서도 리눅스의 활용이 본 궤도에 오르고 전자통신연구원이 주관이 되어 추진 중인 리눅스 클러스터 연구 개

발이 본격화되면 대규모 서버를 위한 분산 병렬 파일 시스템의 필요성이 높아지리라고 본다.

참고문헌

[1] S. Baylor and C. Wu, "Parallel I/O workload characteristics using Vesta." In Proceedings of the IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems, pp. 16-29, 1995.

[2] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J. Prost, M. Snir, B. Traversat, and P. Wong, "Overview of the MPI-IO parallel I/O interface," In Proceedings of the IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems, pp. 1-15, 1995.

[3] S. Moyer and V. Sunderam. "PIOUS: a scalable parallel I/O system for distributed computing environments," In Proceedings of the Scalable High-Performance Computing Conference, pp. 71-78, 1994.

[4] A. Purakayastha, C. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing parallel file-access patterns on a large-scale multiprocessor," In Proceedings of the 9th International Parallel Processing Symposium, pp. 165-172, 1995.

[5] R. Sidebotham, "Volumes: The Andrew File System Data Structuring Primitive," In European Unix User Group Conference Proceedings. August, 1986.

[6] S. Moyer and V. Sunderam, "A Parallel I/O System for High-Performance Distributed Computing," In Proceedings of the IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Distributed Systems, 1994.

[7] M. M. Cettei, W. B. Ligon III, and R. B. Ross, "Support for Parallel Out of Core Applications on Beowulf Workstations", Proceedings of the 1998 IEEE Aerospace Conference, March, 1998.

[8] S. Moyer and V. Sunderam, "Parallel I/O as a Parallel Application," In International Journal of Supercomputer Applications, Vol9, No 2, 1995.

[9] S. Moyer and V. Sunderam, "Scalable Concurrency Control for Parallel File Systems," In IOPADS workshop of the 1995 International Parallel Processing Symposium, 1995.

[10] V. Sunderam, "PVM: A framework for parallel distributed computing," Concurrency: Practice and Experience, Vol. 2, No 4, pp. 315-339, 1990.

[11] A. Tridgell and D. Walsh "The HiDIOS filesystem," .In Fujitsu Parallel Computing Workshop, 1995.

[12] W B. Ligon III and R. B. Ross, "An Overview of the Parallel Virtual File System", Proceedings of the 1999 Extreme Linux Workshop, June, 1999.

[13] Se-Jin Hwang, Jin-Uok Kim, Myong-Soon Park, Oh-Young Kwon, Tae-Geun Kim, "The Prototype of Continuous Media File System, CrownFS", Proceedings of International Conference on Information Networking, pp.235-240, Jan. 1998.

[14] Coda File System, http:/www.coda.cs.cmu.edu/index.html.

[15] The Galley Parallel File System, http://www.cs.dartmouth.edu/~nils/galley.html.

[16] HiDIOS filesystem, http://cafe.anu.edu.au/cap/projects/files.

[17] The PAPERS Project Home Page, http:/garage.

enc.purdue.edu/~papers.

[18] PIOUS for PVM, http://www.mathcs.emory.edu/pious.

[19] The Parallel Virtual File System, http://ede.clemson.edu/parl/pvfs.

[20] ROMIO: A High-Performance, Portable MPI-IO Implementation, http://www-unix .mcs.anl.gov/romio

[21] xFS, http://now.cs.berkeley.edu/Xfs/xfs.html

서 대 화

1981년 경북대학교 전자공학과 (학사)
1983년 한국과학기술원 전산학과 (석사)
1993년 한국과학기술원 전산학과 (박사)

1983년-1995년 한국전자통신연구원
1995년-현재 경북대학교 전자전기공학부 부교수
관심분야 : 분산 및 병렬 컴퓨터 구조, 병렬 운영체제
E-mail : dwseo@ee.knu.ac.kr