파이버채널(Fiber-channel)기반 공유디스크 분산 파일시스템

이 영 무, 김 경 호, 임 상 석, 박 규 호
한국과학기술원 전기 및 전자공학과 컴퓨터공학연구실
Email: {ymlee,kyhkim,sslim,kpark}@core.kaist.ac.kr

Fiber-channel-based Shared Disk Distributed File System

Young Moo Lee, Kyung Ho Kim, Sang Seok Lim, Kyu Ho Park
Computer Engineering Laboratory, Electrical Engineering Department
Korea Advanced Institute of Science and Technology

요 약

파이버채널은 고용량 디스크를 특정 호스트에 의존하지 않고 네트웍에 직접 연결할 수 있는 매우 빠른 입출력 시스템 인터페이스이다. 파이버채널의 등장에 따라 고용량 디스크 어레이를 공유하는 분산 파일시스템에 관한 관심이 늘고 있다. 본 논문에서는 파이버 채널의 장점을 이용하여 효율적인 공유디스크 분산 파일시스템의 설계결과를 제시한다. 제시된 파일시스템은 시스템에 참여하는 모든 호스트들이 부하를 균등하게 나누어 부담하도록 설계되었으며 특히 호스트들이 파이버채널상에서 직접 통신을 통하여 공유디스크 데이타의 일관성을 유지할 수 있도록 하므로써 별도의 장치 없이 일반디스크를 사용할 수 있도록 하여 비용대 성능비가 우수한 시스템에 적합하도록 하였다.

1. 서 론

분산 시스템에서 공유 디스크 모델은 모든 호스트들이 동일한 접근 시간을 가지고 있고 데이타 공유가 용이하다는 장점 있다. 특히 분산 시스템에서 중요한 고장 감내성이 우수하다. 예로서 Digital의 VAX Cluster,Cray Research의 Serverless File System등이 있다.

예전에 소개된 공유 디스크 모델의 시스템들은 확장성 측면에서 문제가 있었으나, 최근 입출력 시스템 인터페이스로 채택되고 있는 파이버 채널을 사용할 경우 디스크들과 호스트들이 별도의 구분없이 네트웍으로 연결될 수 있고 상당히 높은 대역폭을 이용할 수 있어서 문제점을 상당부분 해소할 수 있을 것으로 기대된다.

본 논문에서는 파이버 채널을 이용한 공유 디스크 분산 파일 시스템을 설계함으로써 성능과 가용성이 우수하고 확장이 용이한 분산 파일 시스템을 제시하고자 한다. 파이버채널을 통해 다수의 호스트와 디스크 어레이를 연결하여 구성한 구조위에서 리눅스를 확장하여 설계된 이 시스템은, 특히 부하가 구조적으로 각 호스트에게 균등하게 분배되고, 또한 공유 디스크의 데이타에 대한 동기화측면에서 별도의 장치없이 일반디스크를 사용할 수 있도록 파이버 채널을 이용한 호스크간 직접 통신방식으로 구현하여 비용대비 성능면에서 장점을 가질 수 있도록 하였다.

본 논문의 구성은 다음과 같다. 먼저, 계속 되는 2절에서 제안된 공유 디스크 분산 파일시스템의 구조에 대해서 상세히 살펴보고 3절에서는 현재 구현된 결과를 제시한다. 마지막으로 4절에서 결론과 함께 향후 계획에 대해 소개한다.

2. 시스템 설계

우리가 가정하는 시스템 구성은 arbitrate loop로 구성된 파이버 채널에 하나 이상의 공유 디스크 어레이와 다수의 동일한 호스트 컴퓨터가 연결된 것이다. 이 위에서 리눅스를 확장하여 파일시스템을 구현한다. 설계에 있어서 고려한 특징은 비용대비 성능의 우수성, 부하의 균등분배와 디스크 어레이 데이타의 일관된 공유를 효율적인 방법으로 가능하게 하는 것이다.

전체 디스크 영역은 호스트의 수에 비례하는 지역 영역으로 논리적으로 분할하고 이들을 각각의 호스트에서 관리한다. 이를 관리하는 프로세스를 서버프로세스로 정의한다. 서버프로세스가 관리하는 영역에 대한 접근을 요구하는 프로세스는 그것이 실행되는 호스트에 대한 구분없이 클라이언트프로세스로 정의된다. 동일호스트내에 서버프로세스와 클라이언트프로세스가 공존하며 각 호스트는 자기가 맡은 영역에 대해서 다른 호스트의 클라이언트프로세스에 대해서도 서비스를 제공하는 것이다. 이와 같은 방법으로

공유된 파일에 대한 서비스 부하를 균등분배하고 있다.

사용자에 대한 파일서비스는 파일이 디스크상에 물리적으로 저장된 위치에 관계없이 일정한 경로명으로

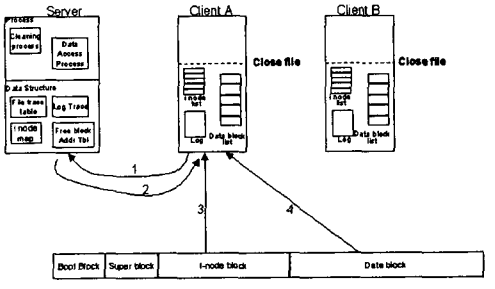


그림 1. 디스크로부터의 파일 읽기

제공되며 데이타의 공유는 파일 단위로 지원되는데

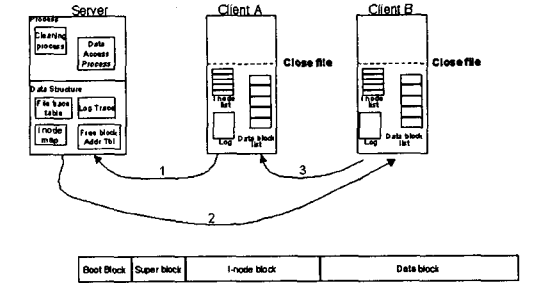


그림 2. 다른 클라이언트로부터 파일 읽기

이 때 수정된 데이타는 파일을 닫을 때 저장되는 방침을 따른다.

각 클라이언트는 다른 호스트를 거치지 않고 직접 파이버 채널상의 모든 디스크에 접근할 수 있다. 그렇기 때문에, 파일의 메타 데이터와 데이터 블록을 직접 읽어 갈 수 있다. 그러나 두개 이상의 호스트가 같은 파일에 쓰기 접근을 할 경우 메타 데이터와 데이터 블록의 단일 동작(atomic operation)이 이루어지지 않으면 파일의 일관성이 유지 될 수 없다. 이와 같이 파일의 일관성을 유지하기 위한 단일 동작을 제공하기 위해서는 락기능을 이용하여 파일 접근이 이루어 져야 한다. GFS[1]에서는 디스크 드라이브내에 메모리를 첨가하여 이와 같은 락기능을 추가하였으나, 이 방법은 특수한 디스크를 요구하고 디스크의 가격을 높일 뿐 아니라 파일의 일관성 유지를 위해서 클라이언트에서 파일 캐쉬를 사용할 수가 없기 때문에 성능이 많이 저하되는 결과를 초래한다. 본 논문에서 제안하는 디자인에서는 락기능이 디스크를 관리하는 서버에 존재하며, 서버는 파일을 연 클라이언트의 상태를 가지고 일관성 유지를 보장 할 수 있도록 설계되어 있기 때문에 클라이언트들은 파일 사용 후 닫을 때 디스크에 곧바로 쓰지 않고 캐쉬할 수 있게 한다.

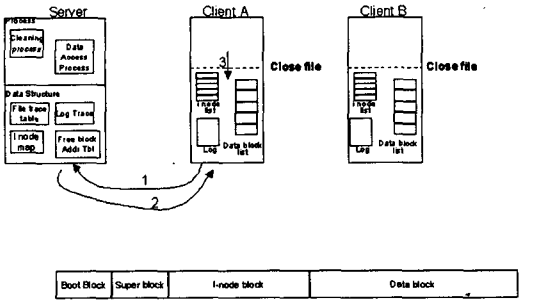


그림 3. 파일 쓰기

또한 캐쉬한 데이터는 일정량이 그 캐쉬의 데이터를 모아서 디스크에 쓸 때 log file system 방식을 사용한다. Log file system은 UNIX file system에 비해 작은 크기의 파일을 쓸 때 좋은 성능을 보이며 크기가 큰 파일을 쓰거나 파일을 읽을 경우에는 비슷한 성능을 보인다.

서버의 구조를 보면 파일 접근 허가, 파일 위치 계산과 log file system을 위한 i-node와 디스크 위치의 함수, 및 디스크의 자유 블록을 관리하는 data access 프로세스와 log file system에서 주기적으로 혹은 시스템이 쉬는 기간에 log의 프레그먼테이션을 정리하는 cleaning 프로세스가 있다. 그러므로 Data access 프로세스를 위해서 inode map, free block list 그리고 file trace table의 스트럭처가 필요하다. 또한 클라이언트는 inode 및 데이터 블록을 갖고 있는 캐쉬와 log 쓰기를 위한 log 버퍼가 존재한다.

클라이언트 A는 서버로부터 그 파일의 접근 허가를 받는다. 읽기 위한 접근은 다른 클라이언트에서 그 파일을 쓰기 위해 가져간 상태를 제외하면 어떠한 경우도 허락된다. 파일의 위치와 함께 접근 허가를 받게 되면 그 파일의 읽기 동작이 끝날 때까지 다른 클라이언트는 사용할 수가 없다. 만약 디스크 상의 파일이 최종 사용된 파일이라면 디스크에서 메타데이터와 데이터 블록을 읽게 된다(그림 1). 파일이 다른 클라이언트 B에서 마지막 씌여진 것이라면 서버에 의해서 클라이언트 B가 그 파일을 클라이언트 A에 전송해 준다. 또한 디스크가 느리기 때문에 디스크 파일과 다른 클라이언트의 캐쉬내의 파일이 같은 최신 version이라면 다른 클라이언트의 캐쉬에서 읽어 가도록 한다(그림 2). 이와 같은 읽기 동작이 끝나게 되면 서버는 그 파일을 다른 클라이언트가 사용 가능하도록 한다. 쓰기의 경우 파일 읽기와 같은 방법으로 파일을 가져간다. 단지 파일을 닫기 전 서버에 먼저 그 파일의 내용이 바뀌었음을 알려주고 난 후 그

파일을 자신의 캐쉬에 쓰고 파일을 닫는다(그림 3).

파일이 닫힌 후 저장된 파일들의 크기가 log 크기에 채워지면 클라이언트는 서버로부터 log 쓰기 허가와 함께 inode map 과 free block list 을 받아서 파일들이 저장될 위치를 정한 후 테이블을 log 를 쓰고 고쳐진 inode map 과 free block list 를 서버에 되돌려 준다(그림 4).

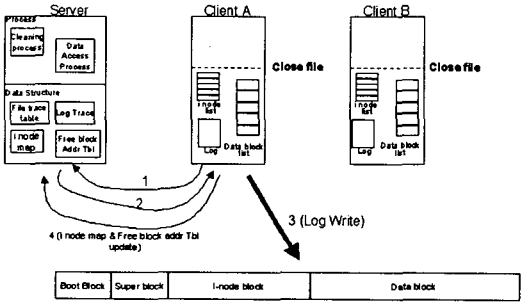


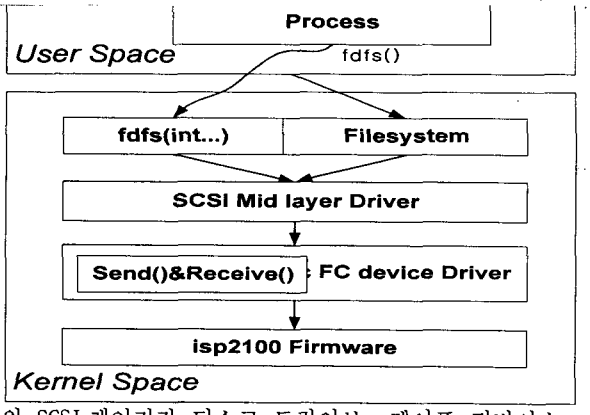
그림 4. Log 쓰기

일반 파일과 달리 디렉토리 파일과 같은 경우는 파일 쓰기의 공유가 유지되어야 한다. 즉 서로 다른 클라이언트가 같은 디렉토리 상에서 작업을 할 때에는 그 디렉토리 파일이 서로 캐쉬되어 있다가 서로 다른 결과를 쓸 수도 있다. 이 경우 디렉토리 파일의 일관성이 유지 되도록 하기 위해 캐쉬는 읽기를 위해서만 사용되도록 하며 쓰기의 경우는 서버로부터 쓰기 허가를 받자 마자 디스크에 쓰도록 하여야 한다. 이 경우 디렉토리 쓰기가 생길 때 마다 항상 디스크 접근이 필요하기 때문에 성능 저하가 예상되나 UNIX 상의 실험[2]에서 디렉토리 쓰기 빈도는 굉장히 낮기 때문에 전체 성능에는 크게 영향을 주지 않을 것으로 예상된다.

3. 시스템 구현

제안된 시스템 구현에 있어서 가장 중요한 구현요소 중의 하나는 SCSI 프로토콜을 사용한 호스트간의 통신을 가능하게 하는 것이다. 대개 입출력 시스템용 어댑터와 그 디바이스 드라이버는 Initiator 로만 동작하기 때문에 이를 target 으로도 동작하도록 하는 것이 필요하다.

리눅스에서 SCSI 프로토콜을 사용하는 파이버채널 호스트 어댑터를 사용하기 위해서는 중간 계층의 SCSI 드라이버를 거치도록 되어 있다. 왜냐 하면 SCSI 제어기는 제조업체에 따라 SCSI 버스의 인터페이스만 같을 뿐 호스트쪽의 인터페이스는 다르기 때문에, 낮은 계층의 서로 다른 SCSI 제어기에 동일한 호스트쪽 인터페이스를 제공하기 위해서 이다. 이 중간 계층의 드라이버는 파일시스템에서 호출하는데 이것은 기존



의 SCSI 제어기가 디스크 드라이브, 테이프 디바이스와 같이 저장용 target 디바이스가 주로 사용되었기 때문이다. 따라서, 이와 같은 구조의 리눅스 SCSI 디바이스 드라이버에 호스트간 통신을 지원하기 위해서 그림 5 와 같이 파일 시스템을 거치지 않고 직접 중간 계층의 디바이스 드라이버를 부를 수 있는 sys_fdfs 라는 시스템 콜을 정의하였고, 이 시스템 콜이 또한 낮은 계층의 디바이스의 send(), receive()를 부른다. 여기서 send(), receive()는 파일시스템에서 오는 디스크 접근에 대한 기존 처리 루틴을 계속 유지해야 한다.

4. 결론 및 추후과제

본 논문에서는 파이버 채널을 이용한 공유 디스크 분산 파일 시스템을 설계하였는데, 파이버채널을 통해 다수의 호스트와 디스크 어레이를 연결하여 구성한 구조위에서 리눅스를 확장하여 설계된 이 시스템은, 특히 부하가 구조적으로 각 호스트에게 균등하게 분배되고, 또한 공유 디스크의 데이타에 대한 동기화 측면에서 별도의 장치없이 일반디스크를 사용할 수 있도록 구현하여 비용대비 성능면에서 다른 시스템보다 우수하다.

현재 디바이스 드라이버수준에서의 호스트간 통신 및 디스크 공유가 구현되었으며 동기화된 공유방법, logging, 파이버채널 밖의 다른 호스트들에 대한 서비스와 같은 상위 수준의 서비스들은 계속 구현중이다.

참고문헌

[1] S. Soltis et al., "The Global File System", in Proc. of the 5th NASAGoddard Space Flight Center Conf. On Mass Storage Systems and Tech., 1996

[2] G. Gibson et al., "File Server Scaling with Network-Attached Secure Disk", in Proc. of the ACM Int'l Conf. on Measurement and Modeling of Computer Systems '97