효율적인 병렬 입출력을 지원하기 위한 분산공유디스크의 설계 및 구현†

송창호°, 남영진, 박찬익
포항공과대학교 전자계산학과

The Design and Implementation of the Distributed Shared Disk for Efficient Parallel I/O

Chang-Ho Song, Young Jin Nam, Chan-Ik Park
Dept. of Computer Science & Engineering, POSTECH

요 약

병렬파일시스템을 분산 환경에서 구현하고자 할 때 하부기능들을 관리 및 유지하기 위해서는 복잡한 내부 동작이 필요하다 저수준의 하드웨어 관리기능들을 고수준의 파일 서비스 기능들과 분리함으로써 병렬파일시스템 구현의 복잡도를 감소시킬 수 있다. 이를 위해 본 논문에서는 분산환경상에서 물리적으로 분산되어 있는 디스크들을 하나의 거대한 논리적인 가상디스크로 보여주는 분산공유디스크 개념을 제안한다. 분산공유디스크는 병렬파일시스템을 지원하기 위한 저수준의 인터페이스를 제공함으로써 병렬파일시스템에서 필요로 하는 하부기능들을 잠재적으로 제공할 수 있다. 또한, 클러스터 기반 시스템에서 분산공유디스크의 프로토타입을 구현하여 그의 동작을 실험하였다.

1. 서론

병렬파일시스템은 병렬 응용에게는 고수준의 병렬 입출력 인터페이스를 제공하고, 내부적으로는 하부 저장장치 및 네트웍 등의 하드웨어와 관련된 복잡한 저수준의 동작을 수행해야 한다. 현재까지 제안되고 구현된 병렬파일시스템들은 설계 및 구현에 있어서 하부 관리 기능과 고수준의 파일 서비스 기능들을 모두 고려해야 한다. 따라서, 병렬파일시스템이 운영하고 있는 내부 메커니즘을 이해하고 새로운 병렬파일시스템을 구현하기 위해서는 많은 시간과 노력이 들어가게 된다.

본 논문에서는 분산환경에서의 병렬파일시스템을 구현하고 이식하는데 드는 비용을 감소시키고자 디스크 및 네트웍 등과 같은 저수준의 하드웨어 관리 기능들을 고수준의 파일 서비스 기능들과 분리시켜서 하부 기능들을 독립적인 단위로 구성하고자 한다. 이를 위해 물리적으로 분산된 비공유 디스크들을 하나의 논리적인 디스크로 보이게 함으로써 하드웨어 수준에서의 추상화를 제공하는 분산공유디스크 개념을 제안하고, 상위 병렬파일시스템에게 제공해야할 인터페이스를 설계하고 이의 프로토타입을 구현하였다.

2. 분산공유디스크의 설계 및 구현

분산공유디스크(Distributed Shared Disk : DSD)는 분산환경에서 물리적으로 분산된 노드들에 존재하는 디스크들을 상위계층에는 하나의 거대한 디스크로 보이도록 네트웍과 저장장치들을 아키텍쳐 수준에서 가상 공유디스크로 추상화한 것이다. 그림 1에 분산공유디스크의 논리적인 뷰가 나타나 있다. 상위계층에서는 분산공유디스크에서 제공하는 인터페이스를 사용함으로써 분산환경하의 비공유 디스크들을 마치 하나의 거대한 지역 디스크로 사용할 수 있게 된다. 그 결과로써 상위 병렬파일시스템은 지역디스크를 운영하는 복잡도만 가지고 쉽게 구현될 수 있으며 잠재적으로 분산공유디스크가 지원해주는 입출력의 병렬성을 얻게 된다. 예를 들어서, MPI-IO[1] 구현시에 현재에는 기존의 파일시스템을 하부계층으로 그대로 이용하든지 새로이 하부 파일시스템 기능을 MPI-IO에서 직접 구현을 해야 한다. 기존의 파일시스템을 이용하는 입장은 시스템에 의존적이고 아키텍쳐수준에서의 병렬성을 지원하지 못한다. 두번째 접근방식은 MPI-IO 자체 내에 파일시스템기능을 포함시켜야 하는 복잡성이 존재하게 된다. 만일 분산공유디스크상에서 MPI-IO를 구현한다면 분산공유디스크가 제공하는 하부기능들을 그대로 이용함으로써 앞서 언급한 두 가지 방법의 모든 문제점을 해결할 수 있다.

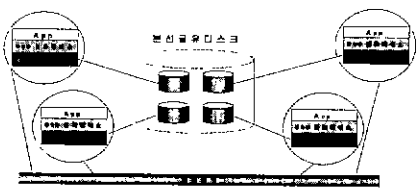


그림 1 분산공유디스크

2.1 분산공유디스크 설계시 고려사항

분산공유디스크를 클러스터 기반 시스템상에서 설계할때에 다음과 같은점들을 고려하여야 한다.

● 논리적-물리적 주소 맵핑

분산공유디스크는 클라이언트에게 물리적으로 분산된 디스크들을 하나의 가상디스크로 추상화함으로써 위치 투명성을 제공한다. 다중 디스크들에 대한 관리를 용이하게 하기 위해서 가상디스크 형태로 추상화하는 것은 몇몇 분산파일시스템에서 사용하였다[2]. 클라이언트는 분산공유디스크가 제공하는 가상 디스크공간상에서 동작을 하게되고, 분산공유디스크에서는 물리적인 디스크 공간을 가상디스크 공간으로 맵핑시켜주는 기능을 가져야 한다. 주소공간 맵핑을 하는 방법은 크게 두가지로 생각할 수 있다 하나는, 간단한 수식을 이용해서

† 본 연구는 과학 기술처 미래원천 기술 개발 사업의 일환으로 수행되었습니다

정적으로 맵핑을 하는 것이다. 이렇게 하면 맵핑자체가 간단할 뿐만 아니라 부가적인 자료구조가 필요없게 되는 장점이 있는 반면 융통성이 떨어진다. 다른 하나는 테이블을 이용하는 방법으로써 동적으로 맵핑을 할 수 있는 반면 부가적인 자료구조를 많이 필요로 하고 복잡한 면이 있다[2].

● 힌트정보를 이용한 파일단위의 스트라이핑
클라이언트는 분산공유디스크에게 블록 배치와 관련된 힌트정보를 제공함으로써 클라이언트가 원하는 대로 블록배치를 할 수 있게 한다. 스트라이핑 패턴(Striping Factor), 스트라이핑 유닛(Striping Unit)크기, 배치방법 등의 힌트정보를 조합해서 클라이언트는 파일단위의 분산배치를 할 수 있다. 이렇게 함으로써 RAID와 같은 디스크 배열 시스템에서 RAID 컨트롤러가 디스크 배열을 관리함으로써 얻어지는 병렬성을 동일하게 지원할 뿐만 아니라, 상위계층에게는 논리적인 하나의 디스크로 보여주게 된다.

● 디스크 입출력 관리
디스크를 관리하는 방법은 크게 두 가지로 생각해 볼 수 있다. 첫째는 중앙서버를 통해서 전체 디스크를 관리하는 방법이 있다. 이 방법은 메타데이타 수정시 일관성유지에 필요한 오버헤드가 크고 병목현상이 발생할 수 있다. 두 번째 방법은 각 서버가 지역적으로 디스크를 운영하는 것이다. 이렇게 되면 메타데이타의 일관성을 유지할 필요가 없을 뿐만 아니라 설계가 간단해진다.

● 확장성
분산공유디스크는 클러스터 기반 시스템을 기초로 한다. 클러스터 시스템은 확장성 있는 (Scalable)네트웍을 기반으로 노드 수의 증가에 따라 그 능력이 비례적으로 증가되는 특성이 있다. 그러나, 여러가지 오버헤드로 인해서 완전히 비례적인 능력을 가질 수는 없다. 따라서, 분산공유디스크는 최대한의 확장성을 보장하기 위한 메커니즘들을 운영하여야 한다. 그러한 측면에서, 캐쉬를 운영하는 것도 확장성을 최대한 지원하기 위한 한 방법이라고 볼 수 있다. 네트웍 성능이 디스크 성능보다 월등히 나은 환경이라면 전역캐쉬를 운영하는것도 효율적이다. 이 이외에 확장성을 최대화 하기위한 여러 가지 방안들에 대해서는 계속 연구해야 할 것이다.

● 집단적 입출력(Collective I/O)
병렬 입출력의 경우 최적화를 위해서 집단적 입출력이 중요하다. 기존의 Two-Phase I/O[5]나 Disk-Directed I/O[6]같은 방법을 분산공유디스크에서 지원하는 것이 가능하며 간단한 형태의 병합(merging), 그룹핑(Grouping)등의 형태로도 지원이 가능하다.

2.2 DSD 구조 및 인터페이스 설계

본 논문에서 구현한 분산공유디스크는 크게 분산공유디스크 인터페이스, 분산공유디스크 스터브 모듈, 분산공유디스크 서버 모듈로 구성된다. 그림 2에 분산공유디스크의 구조가 나타나 있다.

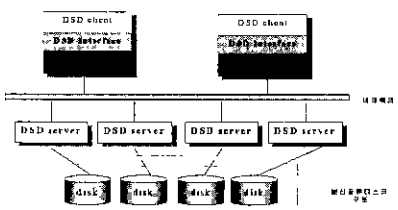


그림 2 분산공유디스크 구조

2.2.1 분산공유디스크 인터페이스

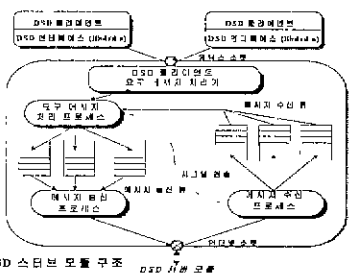
분산공유디스크의 인터페이스는 병렬 파일 시스템이 필요로 하는 기능을 충분히 지원을 하면서 융통성을 최대한 보장하기 위해서 표 1처럼 저장장치 수준에서 기본적인 블록단위의 입출력 서비스를 제공하도록 설계가 되어졌다. 분산공유디스크를 이용하기 위해서는 DSD장치 열기를 수행함으로써 분산공유디스크에 대한 접근권한을 획득한 후 블록 할당, 블록 반환, 블록 읽기, 블록 쓰기 등의 동작을 수행할 수 있다. DSD장치 열기시에 가상 디스크 번호(vdiskId)를 획득하게 됨으로써 클라이언트는 여러개의 가상디스크를 이용할 수도 있다. 임의의 블록 그룹을 할당할때에 블록 배치에 관한 힌트정보를 제공함으로써 블록그룹단위에 대해서 입출력에 대한 병렬성을 제공할 수 있다. 블록 할당시 여러개의 블록들을 한번에 할당 받을 수 있게 함으로써 네트웍 오버헤드를 줄일 수 있고, 비동기 블록쓰기를 지원함으로써 블록 쓰기의 성능을 향상시킬 수 있다. 클라이언트는 블록 싱크(Sync)를 호출함으로써 특정 블록들의 영속성(persistency)을 보장할 수 있다. 또한, 표 1에서 제공하는 인터페이스는 DSD상에서 동작하는 응용프로그램중의 하나인 SIO[4]기능을 구현하기 위해서 충분하다는 것을 검증하였다

표 1 분산공유디스크 인터페이스

기 능	인 터 페 이 스
DSD 장치 열기	int DSD_open(char *devName);
DSD 장치 닫기	dsdResult_t DSD_close(int vdiskId);
블록 할당	dsdResult_t DSD_newBlock(int vdiskId, int blkCnt, dsdBlkId_t *blkSet, dsdLayoutHint_t layoutHint);
블록 반환	dsdResult_t DSD_freeBlock(int vdiskId, int blkCnt, dsdBlkId_t *blkId);
블록 읽기	dsdSize_t DSD_read(int vdiskId, int blkCnt, dsdBlkId_t *blkSet, dsdBuf_t *buf);
블록 쓰기	dsdSize_t DSD_write(int vdiskId, int blkCnt, dsdBlkId_t *blkSet, dsdBuf_t *buf);
블록 쓰기 (비동기)	dsdSize_t DSD_aswc_write(int vdiskId, int blkCnt, dsdBlkId_t *blkSet, dsdBuf_t *buf);
블록 Sync	dsdResult_t DSD_sync(int vdiskId, int blkCnt, dsdBlkId_t *blkSet);

2.2.2 분산공유디스크 스터브

분산공유디스크 스터브 모듈은 클라이언트가 사용하는 논리적 블록번호를 해당 서버와 물리적인 블록번호로 변환하는 기능과 힌트정보를 이용하여 블록을 분산 배치시키는 기능을 가진다. 그림 3에서 보면 DSD 스터브 모듈은 크게 요구 메시지 처리 프로세스, 메시지 수신프로세스, 메시지 송신프로세스 및 클라이언트의 요청메시지를 처리하는 요구메시지 처리기로 구성이 되어져 있다. 요구메시지 처리 프로세스는 클라이언트의 입출력 요구를 각 해당되는 서버의 송신 큐에 분



DSD 스터브 모듈 구조 DSD 서버 모듈

그림 3 DSD 스터브 모듈

할해서 넣는다. 메시지 큐는 서버 개수만큼 존재하고 각 큐마다 하나의 메시지 송신프로세스가 존재해서 큐의 요구메시지를 해당서버로 보낸다. 서버에서 처리된 결과 값들은 수신 프로세스가 받아서 해당 수신 큐에 넣게 되고, 요구 메시지 처리 프로세스에게 시그널을 전송한다. 요구 메시지 처리 프로세스는 자신이 분산해서 보낸 부분요청들이 모두 도착하였을 경우에 각 수신 큐에서 가져와 이를 하나의 결과값으로 구성해서 클라이언트에게 되돌려 주게 된다.

2.2.3 분산공유디스크 서버

분산공유디스크 서버 모듈은 물리적인 디스크가 있는 입출력 노드에 존재하면서 자신의 지역디스크들에 대한 입출력 작업 및 메타데이타를 관리한다. 각각의 서버는 자신에게 오는 요청들에 대해서 지역적으로만 처리를 하면 되고, 서버들끼리 통신을 할 필요가 없다는 점에서 설계 및 구현이 용이하다. 그림 4를 보면 서버 모듈 내에는 메시지 수신 프로세스

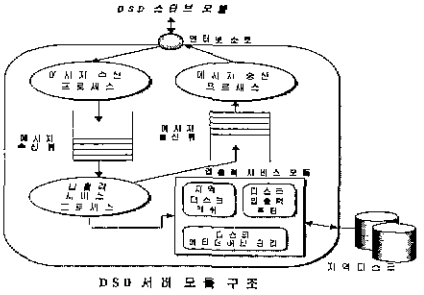


그림 4 DSD 서버 모듈

와 메시지 송신 프로세스가 각각 송신 큐와 수신 큐에 대한 동작들을 수행하게 되고, 입출력 서비스 프로세스는 수신 큐에서 요청 메시지를 꺼내서 해당되는 입출력 서비스를 수행한 후 이를 메시지 송신 큐에 넣어준다. 현재는 FIFO방식으로 큐를 관리하고 있지만 성능향상을 위해서 큐를 효과적으로 관리할 필요가 있다. 그리고, 현재는 서버에 존재하는 캐쉬를 지역적으로 운영하도록 구현되어 있지만 향후 전역적으로 운영할 수 있도록 설계할 계획이다.

2.3 프로토타입 구현

현재 프로토타입은 4대의 머신이 스위치방식의 초고속 넷트웍인 Myrinet[3]으로 연결되어 있는 클러스터 시스템에서 구현되어져 있다. 모든 모듈들은 C 언어를 사용해서 작성되어지 있고, 각 모듈들 내의 송수신 큐들은 공유메모리로 구현이 되어져 있다. 현재 스터브 모듈에는 각 서버에 대해서 각각 송/수신 큐를 관리하고 있고, 서버 모듈에서는 하나의 송수신 큐를 관리한다. 클라이언트와 스터브 모듈들은 유닉스 도메인 소켓을 사용하여 통신을 하고, 스터브 모듈과 서버 모듈은 Myrinet상에서 TCP/IP 프로토콜을 사용하도록 되어져 있다. 현재 비동기 블록 쓰기와 블록 Sync를 제외한 모든 분산공유디스크 인터페이스가 현 프로토타입내에 모두 구현되어 있다.

3. 성능 측정 및 분석

위에서 구현한 DSD 프로토타입의 정상적인 동작 및 확장성을 측정하고자 Myrinet으로 연결된 4대의 웍스테이션상에서 실험하였다. 각 웍스테이션은 25MHz의 microSPARC 프로세서와 32MB의 주기억 메모리, 64KB의 온칩캐쉬를 가지고 있으며 각 웍스테이션내에 동작하는 운영체제는 SunOS 5.4이다. Myrinet은 단방향으로 640Mbit/s의 성능을 내도록 되어 있지만 버스의 클럭속도, 그리고 메모리 속도 등에 의해 실제적인 전송속도가 줄어든다. TCP/IP 프로토콜을 이용할 경우 그 성능이 더욱 저하되는 것으로 알려져 있다. 각 서버는 raw disk를 운영하고 블록 크기는 512B로 되어있으며, 스트라이핑 유닛 크기도 512B로 되어져 있다.

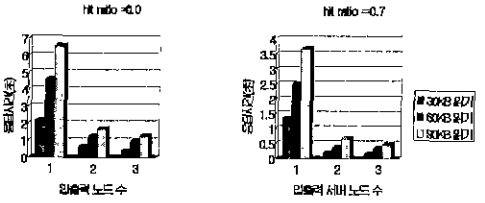


그림 5 입출력 노드수 증가에 따른 응답시간의 변화

실험은 입출력 요구크기를 30KB, 60KB, 90KB로 변화시키면서 입출력 노드수 증가에 따른 응답시간의 변화를 측정하고, 응답시간의 시간분포를 측정하였다. 그림 5에서 보면 입출력 노드수가 증가할수록 응답시간이 짧아짐을 알 수 있고

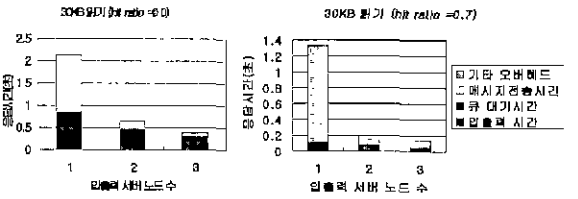


그림 6 응답시간의 시간분포

90KB까지는 입출력 노드수가 2개일 때 응답시간이 포화상태에 도달하는 것으로 나타났다. 서버 캐쉬의 히트율은 임의로 조정하면서 측정을 하였고 서버 캐쉬를 이용해서 전체시간에서 입출력 시간을 줄일 수 있었다. 그림 6은 그림 5의 30KB에 대한 응답시간의 시간분포를 보여주고 있는데 입출력 노드가 1개일 경우 메시지 전송시간이 차지하는 비율이 전체시간의 60%가량을 차지하는 것으로 나타나 있다. 이것은 스터브 모듈과 서버모듈사이에 통신 프로토콜이 TCP/IP의 오버헤드를 고려하지 않았기 때문으로 추정된다. 또한, 입출력 시간이 크게 나타나는 것은 스터브모듈에서 서버모듈로 보내는 요청의 단위가 너무 작아서 지역디스크의 입출력 대역폭을 충분히 활용하지 못했기 때문으로 분석된다. 현재 소스코드가 최적화가 되지 않은 상태이므로 튜닝을 거치면 성능이 크게 향상될 것으로 예상된다. 현재의 구현에서 가장 크게 개선해야 할 부분은 네트웍 오버헤드를 최소화할 수 있도록 큐 관리정책 및 각 모듈간의 통신 프로토콜을 최적화하는 일이다.

4. 결론 및 향후 연구방향

본 논문에서는 병렬파일시스템의 구현 및 이식의 복잡도를 감소시키기 위하여 병렬파일시스템의 하부기능을 독립적인 단위로 분리시킨 분산공유디스크 개념을 제안하고 분산공유디스크의 프로토타입을 구현하였다. 분산공유디스크는 병렬파일시스템 및 MPI-IO의 구현에 드는 비용을 감소시킬 수 있을 것으로 기대된다. 현재 프로토타입의 소스코드가 최적화되지 않았기 때문에 성능이 좋게 나오지는 않았으나 본 실험을 통해서 분산공유디스크의 정상적인 동작을 확인하였다. 향후 분산공유디스크의 각 모듈들을 최적화시키는 일이 가장 시급하다. 분산공유디스크의 성능을 개선하기 위해서는 큐에 있는 입출력 요구들을 모아서 처리하는 등의 효율적인 큐 관리와 네트웍 성능이 굉장히 좋은 환경에서는 전역캐쉬의 운영을 통해서 성능 향상을 얻을 수 있으며 분산공유디스크 모듈간의 통신 프로토콜을 최적화하는 등의 연구가 필요하다. TCP/IP 오버헤드를 개선하기 위해서 Fast Message[7]같은 사용자 수준의 통신 라이브러리를 이용하는 것도 한 방법이다. 그리고, 분산공유디스크 인터페이스가 효율적인 병렬 입출력을 지원하기 위한 저수준의 인터페이스로써 타당한지를 검증하기 위해 분산공유디스크상에 병렬 파일 시스템을 설계하고 구현하는 것도 중요한 향후 연구과제이다.

참고 문헌

[1]*MPI-2.Extensions to the Message-Passing Interface* Message Passing Interface Forum, July 1997.

[2]E Lee and C. Thekkath, "Petal-distributed virtual disks," *Proc 7th Intl. Conf on Architectural Support for Programming Languages and Operating Systems*, pp. 84-92, Oct. 1996.

[3]N.Boden et. al, "Myrinct-a gigabit-per-second local-area network," *IEEE MICRO*, Feb. 1995.

[4]*Proposal for a Common Parallel File System Programming Interface Version 1.0.* Sep. 1996

[5]J Rosario. R. Bordawekar, and A Choudhary, "Improved parallel I/O via a two-phase run-time access strategy," *IPPS'93 Workshop on Input/Output in Parallel Computer Systems*, pp. 56-70, 1993.

[6]David Kotz, "Disk-directed I/O for mimd multiprocessors," *1st Symposium on Operating Systems Design and Implementation(OSDI)*, Nov. 1994.

[7]S. Pakin, V. Karamcheti, and A. Chien, "Fast message: efficient. portable communication for workstation clusters and massively-parallel processors," *IEEE Concurrency*, 1997.