단일 디스크 입출력 환경을 위한 EXT2의 확장

임동혁° 황인철 변은규 맹승렬
한국과학기술원 전자전산학과 전산학전공
{dhlim°, ichwang, ekbyun, maeng}@kaist.ac.kr

Extension of EXT2 for Single Disk I/O Environment

Donghyouk Lim° In-Chul Hwang Eunkyu Byun Sengryoul Maeng
Division of Computer Science, Department of EECS
Korea Advanced Institute of Science and Technology

요 약

단일 디스크 입출력 환경은 클러스터 시스템의 분산된 디스크들을 하나의 통합된 디스크의 이미지로 제공하여 사용자에게 편의성을 제공한다. 하지만, 디바이스 수준에서의 서비스를 제공하고 이로 인해 여러 노드에서의 파일의 병렬적인 접근을 지원하기 위해서는 클러스터 파일 시스템의 지원이 요구된다. 본 논문은 리눅스 시스템에서 가장 많이 사용하는 EXT2 파일 시스템을 단일 입출력 환경에서 효과적으로 사용할 수 있는 클러스터 파일 시스템으로의 확장하는 방안에 대해서 설명한다. 기존의 EXT2 파일 시스템을 커널 모듈의 형태로 재구성하고, 버퍼 캐쉬와 메타 데이터의 일관성 유지를 위하여 분산 락 모듈을 구현하고 이를 이용하여 데이터의 일관성과 동기화 문제를 동시에 해결하도록 하여, EXT2 파일 시스템을 클러스터 파일 시스템으로 확장하였다.

1. 서 론

단일 디스크 입출력 서비스[1]는 클러스터 시스템에서 개별적인 노드들에 분산되어 있는 디스크를 사용자가 더욱 쉽게 사용할 수 있도록 개발된 서비스이다. 단일 디스크 입출력 모듈을 통해서 하나의 통합된 가상 디스크를 제공하고 이것을 이용해서 클러스터 시스템의 사용자는 일반적인 디스크를 사용하는 것처럼 쉽게 사용할 수 있다.

하지만 단일 디스크 입출력은 디바이스 수준에서의 서비스를 제공하기 때문에 한계성을 지니게 된다. 비록 디바이스 수준에서 단일된 이미지를 보여주고 있지만, 실제로 버퍼 캐시 계층은 노드별로 분리되어 있으며, 이로 인해서 파일 시스템 객체에 대한 여러 노드에서의 병렬적인 접근이 이루어지게 되면, 버퍼 캐시에서의 일관성 문제가 생기게 된다. 이러한 일관성 문제를 해결하고 효율적인 단일 디스크 입출력 모듈의 지원을 위해서 새로운 파일 시스템의 구현이 필요하다. 본 논문에서는 이미 기존의 단일 시스템의 리눅스 환경에서 널리 쓰이고 있는 EXT2[2]를 수정 보완하여 클러스터에서 사용할 수 있도록 구현하였다.

또한 클러스터 파일 시스템의 성능을 측정하기 위해서 병렬적인 파일의 접근을 필요로 하는 Equation Solver kernel[7]을 MPI를 이용하여 벤치마크 프로그램으로 작성하고 이를 통하여 파일 시스템의 성능 분석을 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 대표적인 클러스터 파일 시스템과 그 특징에 대해서 논하며, 3장에서는 확장된 EXT2의 구조에 대해서 논한다. 4장은 본 논문의 핵심이 되는 분산 락의 설계에 대해서 살펴보고, 5장은 분산 락을 이용한 EXT2의 확장에 대해서 살펴보고, 6장에서 성능을 분석하고 7장에서 결론을 맺는다.

2. 관련 연구

분산 환경에서의 파일 시스템에 대한 연구는 이미 많은 부분에서 활발하게 진행이 되어 왔다. DEC의 Frangipani[3]는 Petal[5] Virtual Disk라는 고유의 환경을 바탕으로 해서 운영이 된다. 이 시스템에서 제안한 락 시스템은 본 연구에서도 활용이 되었으며, 많은 시스템에서 분산 락을 이용한 일관성 유지 기법을 사용하고 있다. 하지만, 파일 전체 단위의 락을 이용하기 때문에, 병렬적인 쓰기를 지원하지는 못한다.

IBM의 GPFS[6]의 경우도 기본적으로 분산 파일 락을 사용하고 있다. GPFS는 현재 많은 슈퍼컴퓨터에서 파일 시스템으로 채택되어 사용이 되고 있는 고성능의 분산 파일 시스템으로 SAN과 같은 네트워크로 묶인 Shared Storage 환경을 기반으로 하고 있다. GPFS에서는 Frangipani와는 약간 다르게 바이트 단위의 락을 지원하고 데이터와 메타 데이터 모두에 대해서 병렬적인 접근을 지원한다. 메타 데이터의 경우는 락을 가지고 있는 서버가 해당하는 요청들을 모두 다 처리하는 방식을 사용하고 있다.

본 논문의 확장된 EXT2는 앞서 언급한 두 경우의 중간 단계인 블록단위의 락 방식과 흡 기반의 분산 락을 통해서 메타 데이터와 데이터를 분리해서 관리하는 방식의 분산 락을 설계하였고, 이를 통해서 클러스터 파일 시스템으로 확장을 꾀하였다.

3. 확장된 EXT2의 구조

단일 디스크 입출력 환경을 위한 EXT2 파일 시스템의 구조는 다음의 그림 1과 같다. 그림에서 보는 바와 같이 기존의 시스템과는 달리 DL 모듈, 즉 분산 락 모듈이 추가적으로 EXT2 파일 시스템과 버퍼 캐쉬 계층의 사이에 존재하여 둘 사이의 상호작용을 조절한다. 또한 각각의 노드는 디스크가 단일 디스크 입출력으로 묶어져 있을 뿐만 아니라, 분산 락 모듈끼리도 상호협력을 하도록 한다. 분산 락 모듈은 파일 연산에 대한 정보를 모두 다 파일 시스템으로부터 전달 받고 다른 노드들과의 상호협력을 통해서 일관성과 동기화 문제를 동시에 해결해준다.

4. 분산 락 모듈

클러스터 파일 시스템이 기본적으로 갖추어야 할 기능은 파일 시스템의 객체들에 대한 병렬적인 접근을 제어하는 것과 공유하게 되는 데이터의 일관성 유지이다. 데이터의 공유와 함께 동기화 문제가 생기게 되며, 이러한 동기화를 통해서 데이터의 접근이 조절되므로 일관성의 유지 또한 같이 해결할 수 있게 된다. 이 두 가지는 분산 락을 통해서 효과적으로 해결할 수 있다.

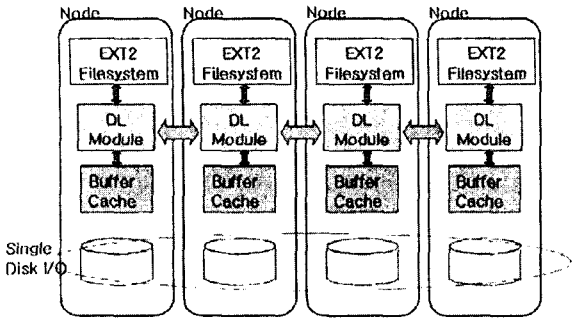

그림 1 단일 디스크 입출력을 위한 EXT2 파일 시스템의 구조

그 이유는 데이터나 메타 데이터에 대해서 여러 노드들 간에 같이 공유되는 부분이 매우 많으며 또한 공유와 함께 해당 데이터의 업데이트도 같이 이루어지게 되기 때문인데, 결국은 동기화와 업데이트는 락 연산과 함께 이루어지게 되고, 락을 풀거나, 쓰기를 하게 되면 이에 맞춰서 업데이트를 실행하도록 하면 그 일관성 문제가 같이 해결이 되게 된다. 이것은 Frangipani[3] 파일 시스템에서의 일관성 관리를 하는 락 서비스와 유사한 방식을 이용한 것이다. 다음과 같은 사항들을 고려하여서 분산 락을 설계 하였다.

4.1 관리 객체의 단위

기본적으로 파일 시스템은 크게 데이터와 메타 데이터의 두 가지로 그 내용을 분리 할 수 있다. 그림 2는 EXT2 파일 시스템의 데이터 구성을 보여주고 있다.[4]

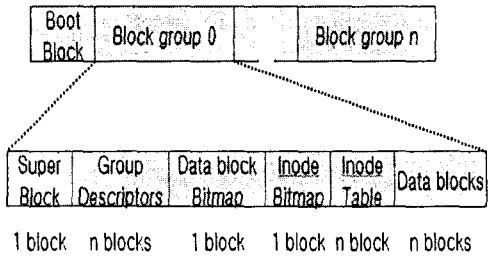

그림 2 EXT2 파일 시스템의 데이터 구성

데이터 블록을 제외한 나머지 것들은 모두 메타 데이터에 속한다. 이러한 객체들은 모두다 디스크에 기록이 되어서 각각은 블록 단위로 관리가 되고, 각각의 버퍼 캐쉬에 저장이 되어서 성능 향상을 돕는다.

단, 슈퍼블록과 아이노드의 경우는 각각이 운영체제에서 제공하는 파일 시스템의 일반적인 인터페이스인 VFS에서 따로 객체를 만들어서 보관을 하게 된다. 결국은 나머지 메타 데이터는 데이터 블록과 동일한 취급을 받게 된다. EXT2는 VFS 아이노드, 슈퍼블록을 수시로 참조하고 메모리에 한번 로드가 되면 VFS 객체만을 참조해서 많은 일들을 수행하게 된다. 실제로 일관성 유지가 필요한 것은 메모리에 올라와 있는 구조체인 VFS 아이노드와 각각의 블록들을 캐쉬하고 있는 버퍼 캐쉬가 된다. 그러므로 분산 락 파일 모듈에서는 VFS 아이노드와 버퍼 캐쉬를 락의 대상 객체로 삼고 동기화와 일관성을 맞추어 주었다.

4.2 락 관리자

각각의 객체에 해당하는 락과 그 정보 들을 관리하는 관리자의 위치는 성능에 매우 중요한 영향을 끼치게 된다. 각각의 객체에 대한 모든 연산과정에서 개입하게 되는 것이 락 관리자이기 때문이다. 기본적으로 객체의 락 관리자(홈)를 할당하는 방식은 해쉬를 이용한 분산을 선택하였다. 클러스터 시스템은 비교적 노드의 참가와 이탈이 적고 구성이 상당히 정적인 특성을 띄므로 쉽게 구현할 수 있으며, 아이노드와 블록의 번호 역시 비교적 고르게 분포되어서, 각 노드당 담당하는 객체의 수는 고른 분포를 띄게 되어서 메시지의 집중화로 인한 병목 현상을 비교적 줄여줄 수 있다.

4.3 락 연산

락 연산에는 기본적으로 lock, unlock의 두 가지 연산이 있다. 분산 락에서는 두 가지의 기본적인 연산을 원래의 의미대로 동일하게 구현하였다. 그리고 파일 시스템에서의 필요에 의해서 원자성이(Atomic) 있는 읽기와 쓰기 연산도 역시 추가가 되었다. 각각의 읽기/쓰기 연산은 블록과 아이노드에 대해서 따로 구현이 되어 있다. 또한 lock 연산과 별도로 해당하는 블록과 아이노드의 최신 복사본을 가지고 업데이트를 하기 위해서 각각의 블록, 아이노드에 대해서 업데이트 연산도 추가하였다.

4.4 일관성 유지 프로토콜

객체에 대한 연산의 결과는 분산 락의 해제와 함께 반영이 되면 된다. 이러한 사실에 기인해서, 크게 네 가지 연산에 대해서 표 1과 같은 작업을 수행하면 된다.

데이터의 유지 방법에 있어서 업데이트와 무효화(Invalidate)방식의 두 가지의 선택사항이 있다. 큰 데이터에는 무효화 방식이, 그리고 작은 데이터에는 업데이트 방식이 좋은 성능을 나타내준다. 블록의 사이즈는 1KB~4KB이고, 아이노드 데이터의 경우는 100byte 정도의 크기가 되므로 비교적 작은 양의 데이터이므로 업데이트 방식을 사용하여서 최신의 데이터를 유지하도록 하였다. 그리고 데이터의 전송은 객체의 홈 노드가 전담해서 데이터의 업데이트가 모두 끝난 다음에 unlock이나 write 연산이 마치도록 구현하였다.

표 1 락 연산과 수행 작업

연산의 종류	수행해야 하는 작업
Lock	노드에서 공유자원을 처음으로 참조할 때 최신의 복사본을 넘겨준다.
Unlock	다른 노드들을 전부 Update
Read	노드에서 공유자원을 처음으로 참조할 때 최신의 복사본을 넘겨준다.
Write	다른 노드들을 전부 Update

5. 분산 락을 이용한 EXT2의 확장

5.1 모듈화

EXT2는 리눅스에서 가본적으로 사용하는 파일 시스템으로 대다수의 커널 소스에 포함되어 있는 상태이다. 개별적인 파일 시스템으로 분리해내고 기존의 시스템에 대한 수정이 필요없이 이식성을 좋게 하기 위해서 커널 모듈의 형태로 분리를 해냈다.

5.2 분산 락의 사용

앞에서 구현된 분산 락을 이용해서 EXT2를 확장시키기 위해서는 파일 및 디렉토리 연산에 대해서 일단 분석을 하고 각각의 과정에서 일관성 유지와 동기화에 대해서 고려해 주어야 한다.

가장 기본이 되는 연산은 각각 readpage, writepage, prepare_write, commit_write 의 네 가지 연산이 바로 그것인데, 이 연산들은 파일과 디렉토리에 대한 연산에서 같이 사용되는 가

장 기초가 되는 연산이므로 이 연산들에서 일관성을 유지를 해주게 되면 다른 연산들에서도 대다수가 해결이 되었다.

일관성과 함께 동기화가 가장 중요시 되는 부분이 전체 파일 시스템 정보를 담고 있는 수퍼블록에 대한 연산이다. 각각 read_super, write_super가 되는데 이 부분에서 동기화에 해당하는 부분에 맞추어서 수퍼블록에 해당하는 버퍼캐쉬에 락 연산을 사용해서 전체적인 동기화와 함께 수퍼블록의 데이터 일관성 문제도 해결하였다.

아이노드의 경우에도 read_inode, update_inode 함수에서 각각을 분석하고 아이노드 락 연산을 사용해서 동기화와 데이터의 일관성 문제를 해결해주었다.

6. 성능 평가

본 확장된 EXT2 파일 시스템의 평가를 위해서 병렬성을 지니는 벤치마크 프로그램을 직접 작성하여서 파일 시스템의 성능을 측정하였다. 본 클러스터 파일 시스템에서 대조군으로 사용한 것은 PVFS[8]로, 클러스터 파일 시스템으로 많이 사용되고 있는 파일 시스템이다. 실험을 수행한 환경은 다음과 같다.

표 2 실험 환경

CPU	Pentium IV 1.8GHz
Memory	512MByte 266MHz DDR Memory
Disk	IBM 60G 7200rpm
Network	3c996B-T(Gigabit Ethernet) 3c17701-ME(24port Gigabit Ethernet Switch)
PVFS	PVFS 1.6.0
Linux	RedHat 9.0(Kernel 2.4.20)

실험에 사용한 벤치마크 프로그램은 256*256 행렬의 내용을 읽어서 4개의 노드에서 분배해서 읽고 유체의 흐름을 시뮬레이션 하는 Equation Solver kernel[7]을 구현하여 전체적으로 20라운드의 작업을 수행하고 행렬을 읽고 쓰는데 걸리는 시간을 누적하여 측정하여 비교하였다. 그림 3은 그 결과를 도표로 나타낸 것이다. 그림 3의 NExt2 는 새롭게 확장한 EXT2를 지칭한다.

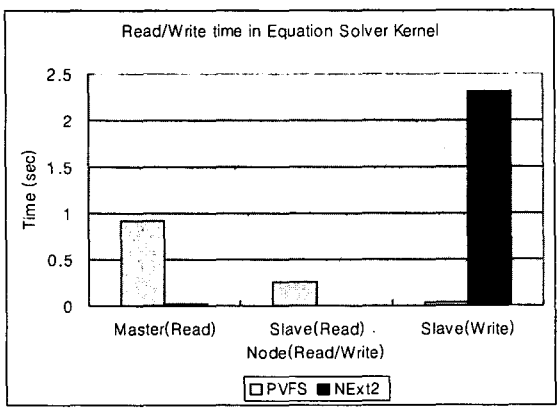

그림 3 Equation Solver Kernel 에서의 수행시간

그림의 결과에서 볼 수 있듯이 확장된 EXT2는 읽기 성능에서는 PVFS에 비해서 상당히 좋은 성능을 보여준다. 반면에 쓰기 성능은 매우 저하되는 것을 볼 수 있다. 이것은 EXT2의 확장을 위해서 구현한 분산 락 모듈의 특성 상 이러한 결과를 가져오게 된 것이라 할 수 있다. 업데이트 방식의 프로토콜을 지원하게 되므로 쓰기과정에서는 각 노드에 최신의 복사본을 전달하고 그 갱신을 확인하는 절차가 이루어지게 되는데 이로 인해서 쓰기 성능은 저하되지만, 업데이트로 인해 읽기 연산을 수행할 때는 흉 노드의 개입이 최소화 되어, 더욱 좋은 성능을 보여주게 된다. 쓰기 성능의 양보를 통해서 읽기 성능이 더욱 좋아지게 되는 것이다. 반면, PVFS의 경우 상호협력 캐쉬가 구현되어 있지 않고, 이로 인해서 각각의 연산마다 I/O 데몬을 통해서 항상 그 결과를 전송받게 되므로 읽기는 상당히 느려질 수밖에 없다. 하지만 쓰기의 경우는 I/O 데몬으로의 전송만을 통해서 완료되어서 비교적 좋은 성능을 보이게 된다.

현재의 일관성 유지 모델와 분산 락의 프로토콜의 개선이 이루어진다면 쓰기 성능의 향상을 통해서 더 좋은 성능을 기대할 수 있으리라 생각한다.

7. 결론 및 향후 과제

단일 디스크 입출력을 이용해서 사용자는 통합된 디바이스를 얻을 수 있지만, 그 실질적인 이용을 위해서는 클러스터 파일 시스템의 이용이 요구된다. 본 논문에서는 널리 사용하고 있는 분산 락을 모듈로 구현하고, 이를 이용하여 기존의 단일 시스템에서 널리 사용하고 있는 EXT2 파일 시스템을 확장하여 클러스터 파일 시스템으로 구현하였다.

성능 평가 결과 PVFS에 비해서 읽기 성능은 상당히 우수하나 쓰기 성능이 매우 저조한 결과를 보여주었다. 이는 분산 락의 설계에 기인한 문제점으로 업데이트 방식으로 인한 쓰기의 오버헤드와 읽기의 반응시간의 Tradeoff로 볼 수 있다.

이를 해결하기 위한 일관성 모델의 연구와 프로토콜의 개선이 이루어진다면 쓰기 성능의 큰 저하를 효과적으로 처리 할 수 있을것이라 기대한다. 또한 EXT2의 구조에 기인하는 대용량 파일의 지원 및 성능 향상 또한 해결된다면 효율적인 파일 시스템이 되리라 생각한다. 본 연구를 통해서 구현된 분산 락 모듈은 캐쉬 계층의 추가와 함께 좀더 일반화를 통해서 상호협력 캐쉬의 모듈로 발전해 나갈 수 있으리라 기대된다.

8. 참고 문헌

[1] 황인철, 김동환, 김호진, 맹승렬, 조정완, "단일 디스크 입출력을 위한 커널 모듈 프로토타입의 설계 및 구현", 한국정보과학회 추계 학술발표논문집, 2003. 10.

[2] Remy Card, Theodore Ts'o, and Stephen Tweedie. "*Design and implementation of the Second Extended Filesystem*", First Dutch International Symposium on Linux, Dec. 1994.

[3] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani : A Scalable Distributed File System", Symposium on Operating Systems Principles, 1997.

[4] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel", O'Reilly, 2003.

[5] E. K. Lee and C. A. Thekkath, "Petal: Distributed Virtual Disks", The International Conference on Architectural Support for Programming Languages and Operating Systems, 1996.

[6] F. Schumuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", The conference of File And Storage Technologies, Jan. 2002

[7] D. Culler and J. Singh, "Parallel Computer Architecture : A Hardware/Sofware Approach", Morgan Kaufmann Publishers Inc.

[8] P. Carns, W. Ligon III, R. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters", Annual Linux Showcase and Conference, 2000.