PVFS를 위한 상호협력 캐쉬의 설계 및 구현

황인철° 김호중 정한조 김동환 김호진 맹승렬 조정완
한국과학기술원 전자전산학과 전산학전공
{ichwang°, hjkim, hanjo, dhkim, Hojin, maeng, jwcho}@camars.kaist.ac.kr

The Design and Implementation of the Cooperative Cache for PVFS

In-Chul Hwang° Hojoong Kim Hanjo Jung Dong-Hwan Kim Hojin Ghim
Seung-Ryoul Maeng Jung-Wan Cho

Division of Computer Science, Dept. of Electrical Engineering & Computer Science
KAIST

요 약

요즘 값싼 PC들을 빠른 네트웍으로 묶어 높은 성능을 얻고자 하는 클러스터 컴퓨팅에 대한 연구가 활발히 이루어지고 있다. 이러한 연구 중 파일에 대한 서비스를 제공하여 주는 파일 시스템에서 높은 대역폭과 병렬성을 지원하는 분산 파일 시스템이 개발되고 있다.

한편 기존 분산 파일 시스템에 대한 연구 중 서버의 부하를 감소시키고 성능을 향상시키기 위하여 상호협력 캐쉬가 제시되었다. 상호협력 캐쉬는 클라이언트간 파일 캐쉬를 공유함으로써 자신에게 없는 파일에 대한 내용을 다른 클라이언트가 가지고 있을 경우 서버에게 파일을 요구하지 않고 클라이언트간 파일 내용 전달을 통하여 요구를 처리하게 된다.

분산 파일 시스템 중 클러스터 컴퓨팅에서 많이 사용되고 있는 Linux 운영체제에서 구현된 PVFS는 높은 성능과 병렬 I/O를 제공한다. 하지만 기존 PVFS에서는 파일에 대한 캐쉬를 제공하지 않는다. 따라서 본 논문에서는 기존 PVFS에서 제공하지 않은 상호협력 캐쉬를 설계하고 구현한다. 그리고 기존 PVFS와의 성능 비교를 통하여 캐쉬의 효율성을 증명한다.

1. 서론

요즘 값싼 PC들을 빠른 네트웍으로 묶어 높은 성능을 얻고자 하는 클러스터 컴퓨팅에 대한 연구가 많이 이루어지고 있다. 클러스터 컴퓨팅을 효율적으로 하기 위해서는 효율적인 네트웍의 구성과 운영체제의 효율적인 서비스가 필요하다. 이러한 연구의 한 분야로서 클러스터 컴퓨팅에서 각 노드의 CPU나 메모리에 비하여 상대적으로 느린 디스크에 접근하는 파일 시스템을 효율적으로 구성하려는 연구가 이루어지고 있다.

한편 기존 분산 파일 시스템에서 파일 서버의 부하를 감소시키고 시스템 전체의 성능을 높이고자 하는 상호협력 캐쉬(cooperative cache)[4,5,6]가 제시되었다. 네트웍을 통한 다른 클라이언트의 메모리에 접근하는 시간이 서버의 디스크에 접근하는 시간보다 빠르다는 점을 이용하여 클라이언트간의 상호협력을 통한 캐쉬를 구성하고자 하는 방식이다.

기존 분산 파일 시스템 중 클러스터 컴퓨팅에서 많이 사용하는 운영체제인 리눅스에서 병렬 I/O를 제공하는 PVFS(Parallel Virtual File System) [1,2]가 개발되었다. PVFS는 파일 데이터 분산을 통하여 동시에 I/O서버들에게 서비스를 받아 높은 대역폭을 제공한다. 하지만 PVFS에서는 파일 캐쉬를 제공하지 않으며 단순히 사용자 프로그램에게 파일의 내용을 I/O 서버로부터 전달해 주는 서비스 만을 제공한다. 따라서 본 논문에서는 높은 성능을 얻기 위하여 PVFS를 위한 상호협력 캐쉬를 설계하고 구현한다. 그리고 구현된 상호협력 캐쉬를 사용한 PVFS와 기존 PVFS를 비교 평가한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구로서 PVFS와 상호협력 캐쉬에 대하여 설명한다. 3장에서는 PVFS를 위한 상호협력 캐쉬의 설계와 구현에 대하여 설명한다. 4장에서는 구현된 시스템과 기존 PVFS와의 성능을 평가하고 비교한다. 5장에서는 향후 연구 방향과 결론을 맺는다.

2. 관련연구

2.1 PVFS(Parallel Virtual File System)

PVFS[1,2]는 Linux 운영체제에서 병렬 I/O를 제공하기 위하여 Clemson University에서 구현한 파일 시스템이다.

PVFS는 크게 계산 노드와 관리자 노드, 그리고 I/O 서버로 구성이 된다.

PVFS의 사용자 접근 방법을 살펴보면 두가지 방법이 존재한다. 첫번째 방법은 사용자 라이브러리를 이용한 방법으로 새로 정의된 라이브러리를 이용하여 프로그램을 재구성하는 방법이다. 다른 한가지 방법은 클라이언트에 PVFS 커널 모듈을 사용하여 기존 UNIX I/O API와 같은 방법으로 접근하는 방법이다.

Vilayannur[3]는 기존 PVFS에서 라이브러리를 이용한 방법을 개선하여 PVFS를 위한 단일 노드 캐쉬 시스템을 구현하고 한 노드에서 여러 프로그램이 데이터를 공유할 경우를 가정하여 캐쉬의 효율성을 보여주었다. 그러나 실제 클러스터 환경에서는 여러 사용자가 여러 노드에서 같은 파일에 대한 접근을 하기 때문에 본 논문에서는 Vilayannur와 달리 단일 사용자가아닌 여러 사용자가 커널 모듈을 이용할 경우를 가정하여 상호협력 캐쉬를 커널 모듈을 수정함으로써 구현하였다.

2.2 상호협력 캐쉬

상호협력 캐쉬[4]는 기존 요구 처리 단계 중 클라이언트가 자신의 요구가 자신의 캐쉬에서 처리가 되지 않을 경우 서버에게 요청하기 전 그 블럭을 캐싱하고 있는 다른 클라이언트에게 그 블럭에 대한 요청을 하여 자신의 요구를 처리하는 방법이다.

이러한 상호협력 캐쉬에 대하여 많은 연구가 이루어 졌다. Dahlin[4]은 캐쉬 블럭들의 관리를 위하여 N-chance 알고리즘을 제안하였고, Feeley[5]는 GMS(Global Memroy Service)상에서 효율적인 캐쉬 블럭 알고리즘을 제안하였다. 그리고 Sarkar[6]는 기존 상호협력 캐쉬에서 정확한 클라이언트 캐싱 정보를 가지고 있던 것을 단순한 힌트에 의해 블럭의 캐싱 정보를 유지함으로서 캐싱 정보를 유지하는데 필요한 부하를 줄이는 방법을 제안하였다.

그러나 Sarkar의 방법은 파일 단위의 일관성을 유지하기 때문에 큰 파일을 여러 노드의 사용자들이 공유하는 병렬 파일 시스템에서는 어울리지 않는다는 단점이 있다.

3. PVFS를 위한 상호협력 캐쉬의 설계와 구현

기존 커널 모듈을 이용한 PVFS에 본 논문에서 제시한 상호협력 캐쉬를 추가하여 설계한 모습은 다음 [그림 1]와 같다.

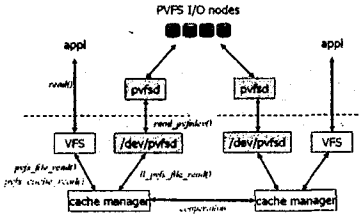

그림 1 캐쉬 관리자가 추가된 PVFS 클라이언트의 구조

[그림 1]에서와 같이 읽기 요구가 들어왔을때 기존과 달리 자신의 캐쉬를 먼저 살펴보고 존재하는 내용은 바로 처리하도록 하고, 자신의 캐쉬에 데이터가 없을 경우 다른 클라이언트의 캐쉬 관리자에게 데이터를 요청하여 처리한다. 그리고 요청이 다른 클라이언트 캐쉬에도 데이터가 존재하지 않을 경우에만 I/O 서버에게 그 내용을 요청하게 된다.

3.1 PVFS를 위한 상호협력 캐쉬의 설계

다음 [그림 2]는 PVFS를 위한 상호협력 캐쉬의 구조를 나타낸다.

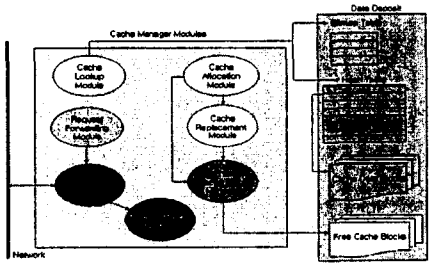

그림 2 상호협력 캐쉬의 구조

[그림 2]에서의 각 모듈은 다음과 같은 동작을 수행한다.

- 캐쉬 찾기 모듈(Cache lookup module) : 요구가 들어오면 캐쉬 블럭이 있는지 해쉬 테이블을 통하여 찾아본다.
- 캐쉬 할당 모듈(Cache allocation module) : 새로운 캐쉬 블럭을 할당한다. 유휴 캐쉬 블럭, 시스템 메모리, 캐싱되어 있던 블럭의 순서로 캐쉬를 할당한다.
- 캐쉬 대체 모듈(Cache replacement module) : 메모리가 부족해졌을 경우 일정 시간마다 깨어나서 캐싱되어있던 블럭들을 해제한다.
- 유휴 캐쉬 블럭 관리자(Free cache block manager) : 시스템으로부터 메모리 할당이 되었으나 사용되지 않은 블럭들을 관리한다.
- 요청 전달 모듈(Request Forwarding Module) : 캐쉬 관리자로부터 요청을 받아서 다른 클라이언트나 I/O서버에게 요청을 전달하여 처리한다.
- 통신 모듈(Communication module) : 통신 모듈에서는 자신의 캐쉬에 있지 않은 데이터를 다른 클라이언트에게 요청하는 메시지를 전달하고 이에 따른 데이터를 전달 받는다. 또한 다른 클라이언트로부터의 데이터 요청을 받아들여 그 요청을 다른 클라이언트로부터 온 요청 처리 모듈에게 전달한다.
- 다른 클라이언트로부터 온 요청 처리 모듈(Remote request handler module) : 다른 클라이언트로부터 요청받은 데이터를 자신의 캐쉬에 있는지 찾아보고 있으면 전달한다.

3.2 PVFS를 위한 상호협력 캐쉬의 구현

상호협력 캐쉬 관리자는 3.1절에서 설계된 것을 기반으로 다음 [그림 3]과 같이 구현하였다.

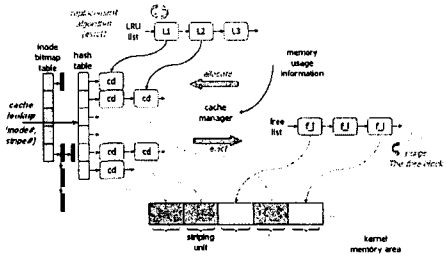

그림 3 상호협력 캐쉬의 구현

[그림 3]에서 보는 바와 같이 모든 캐싱된 블럭들은 연결 리스트를 사용하여 구현되었고, 캐쉬 블럭 대체를 위하여 LRU 리스트를 관리한다. 그리고 할당되었지만 사용되지 않은 블럭들을 유휴 리스트에 관리한다. 캐싱된 블럭은 하나의 분산단위(striping unit)을 기본으로 관리된다. PVFS에서는 분산단위를 기본 64KB로 하며, 시스템에 적합하게 설정이 가능하다. 이러한 캐쉬 블럭이나 LRU 리스트, 유휴 리스트, 해쉬 테이블, 비트맵 테이블들은 많은 모듈에서 동시 접근 가능하기 때문에 일관성을 위하여 락(lock)을 사용하여 동시성을 추구하였다.

본 논문의 상호협력 캐쉬는 기존 리눅스 버퍼 캐쉬 영역을 할당받아 사용할 경우 PVFS 캐쉬 관리자는 시스템에서 임의적으로 캐쉬 블럭을 해제하면 자신이 가지고 있지 않는 블럭에 대한 정보의 갱신을 할 수 없기 때문에 리눅스 버퍼 캐쉬를 사용하지 않는다. 따라서 상호협력 캐쉬에서는 시스템의 메모리 사용량에 따라 캐쉬 블럭의 양을 관리해야 한다. 이를 위하여 캐쉬 대체 모듈은 커널 쓰레드[7]로 구현되어 일정한 시간마다 깨어나서 시스템 메모리의 양을 확인하여 시스템 메모리의 양이 일정량 이하로 줄어들면 캐싱되어 있던 블럭들을 해제시킨다.

상호협력 캐쉬를 동작시키기 위해서는 캐쉬 관리자가 다른 클라이언트가 캐싱하고 있는 데이터의 정보를 가지고 있어야 한다. 본 논문에서는 관리 비용이 싸고 확장성이 뛰어난 힌트를 기반으로 하는 상호 협력 캐쉬를 설계하고 구현하였다. 그리고 병렬 I/O를 지원하는 파일 시스템에 맞지 않는 기존 힌트를 이용한 방법과 달리 본 논문 설계하고 구현한 상호협력 캐쉬에서는 병렬 I/O 파일 시스템에 적합한 블럭단위의 일관성을 유지한다.

힌트를 이용한 캐쉬 블럭 정보를 유지하기 위하여 다음과 같은 방법을 이용한다. PVFS에서는 사용자가 처음 파일을 열 경우 메타데이터 관리자로부터 자신이 열 파일의 메타데이터를 가져와야하는데 이를 이용하여 메타데이터 관리자에서 기존에 파일을 열었던 모든 클라이언트들의 정보를 유지한다. 새로운 클라이언트가 파일을 열 때 메타데이터와 함께 기존 파일을 열었던 클라이언트들의 정보를 가지고 온다. 자신의 캐쉬에 없는 블럭을 읽으려고 할 경우 기존에 파일을 열었던 다른 클라이언트에게 그 클라이언트가 지니고 있는 데이터 블럭에 대한 정보 요청과 함께 자신이 가지고 있는 데이터 블럭의 정보를 보내고, 그 요청을 받은 클라이언트는 데이터 블럭에 대한 정보를 보내주게 된다. 따라서 파일 데이터 블럭에 대한 클라이언트들의 정확한 정보를 유지하지는 않지만 파일 데이터를 많이 접근할 경우 서로 자신의 데이터 블럭 정보를 주고 받게 되어 상호협력을 할 수 있다.

기존 PVFS와 같은 일관성을 유지하기 위해서는 데이터에 대한 쓰기 요구가 요청되었을 경우 캐싱되어 있는 블럭을 모두 해제시켜야 한다. 따라서 쓰기 요구일때 메타데이터 관리자에게 캐싱되어 있던 블럭을 해제하라는 요청을 하게 되고, 이 요구를 받은 메타데이터 관리자는 기존에 파일을 열었던 모든 클라이언트에게 블럭 해제 요청을 함으로써 기존 PVFS와 같은 일관성을 유지시킨다.

4. 성능 평가

성능 평가를 위하여 사용된 시스템은 다음 [표 1]과 같다.

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)

표 1 시스템 사양

메타데이터 관리자와 I/O 서버는 각각 하나의 노드에 할당하였고 네개의 노드에 클라이언트를 할당하여 다음 프로그램들을 수행하였다.

- 읽기/쓰기 프로그램 : 읽기/쓰기 요구 크기를 변화시켜 가면서 읽기/쓰기를 수행한다. 읽기일 경우 서버에 캐쉬이 되어 있는 경우(cache_hot_coop), 다른 노드에 캐쉬이 되어 있는 경우(coop_hot_coop), I/O서버의 캐쉬에 있는 경우(io_hot_coop)/없는 경우(io_cool_coop), 기존 PVFS에서 I/O서버의 캐쉬에 있는 경우(io_hot_pvfs)/없는 경우(io_cool_coop)에 따라 여러 번 수행하여 평균값을 계산하였다. 쓰기의 경우에는 캐쉬하고 있는 다른 클라이언트가 있을 경우(cli_ex)/없을 경우(no_cli), 기존 PVFS(pvfs)에 따라 여러 번 수행하여 평균값을 계산하였다.
- 행렬 곱셈 프로그램 : 파일에 저장되어 있는 1024*1024 행렬 두개를 읽은 후 곱하여 그 결과를 파일에 적는 프로그램을 수행한다. 네개의 클라이언트에서 MPI로 수행되도록 하였다.

4.1 읽기 수행 결과

다음 [그림 4]는 위 여섯가지 경우를 읽기 블럭의 크기를 변화 시켜가면서 읽기를 수행한 수행시간을 나타낸 결과이다.

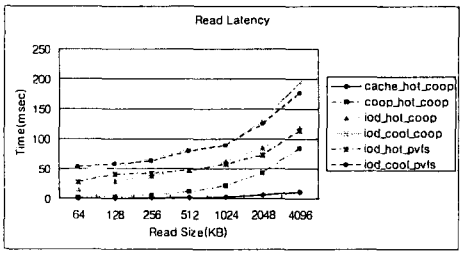

그림 4 읽기 수행 결과

[그림 4]에서 측정한 결과처럼 캐쉬 관리자를 사용한 읽기 성능은 기존 PVFS를 사용하는 것 보다 캐쉬을 사용하는 것이 더 좋은 성능을 나타낸다. 그리고 상호협력 캐쉬를 이용하는 것이 I/O 서버에서 캐쉬이 되어 있는 경우보다 더 좋은 성능을 나타낸다.

4.2 쓰기 수행 결과

각 경우에 대하여 쓰기 수행 결과는 다음 [그림 5]와 같다.

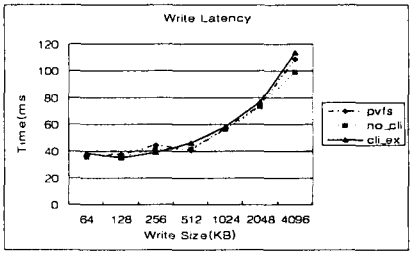

그림 5 쓰기 수행 결과

[그림 5]에서 살펴본 바와 같이 쓰기의 경우는 기존 PVFS에 비해 상호협력 캐쉬 관리자를 사용하는 것이 거의 유사한 성능을 나타 낸다. 이런 결과는 본 논문에서 설계하고 구현한 상호협력 캐쉬 관리자에서 쓰기에 대한 버퍼링을 수행하지 않기 때문이다.

4.3 행렬 곱셈 프로그램 수행 결과

행렬 곱셈 프로그램을 수행한 결과 표 2과 같은 결과를 얻었다.

기존 PVFS의 경우	124.137667초
상호협력 캐쉬를 이용할 경우	119.071566초

표 2 행렬 곱셈 프로그램 수행 결과

같은 파일에 읽기/쓰기를 수행하는 간단한 행렬 곱셈 프로그램에서 [표 2]와 같이 기존 PVFS에 비해 상호협력 캐쉬를 이용할 경우 약 4%정도의 수행시간 향상을 얻을 수 있었다.

5. 결론 및 향후 과제

본 논문에서 클러스터 컴퓨팅에서 많이 사용되는 Linux 운영체제를 지원하는 병렬 파일 시스템인 PVFS에서 지원하지 않던 상호협력 캐쉬를 설계하고 구현하였다.

읽기 요구의 크기를 변경하면서 여러 환경에서 기존 PVFS와 비교 평가하였을 경우 캐쉬를 사용하는 것이 사용하지 않는 시스템보다 빠른 응답시간 안에 요구를 해결할 수 있었다. 쓰기 요구의 크기를 변경하면서 여러 환경에서 기존 PVFS와 비교하여 보았을 때 큰 차이 없이 유사한 성능을 나타내었다. 그리고 간단한 행렬 곱셈 프로그램에서 기존 PVFS보다 약 4%정도의 성능향상을 보였다. 성능 평가에서 살펴본 바와 상호협력 캐쉬를 사용하는 PVFS를 사용할 경우 대용량 파일을 접근하는 여러 응용프로그램들의 성능이 개선되고 클러스터 전체의 성능 향상에 큰 도움이 될것을 추측할 수 있었다.

향후에는 실제 클러스터 시스템을 사용하는 응용프로그램을 가지고 성능평가와 성능분석을 할 예정이고, 쓰기의 경우 높은 성능을 얻기 위하여 버퍼링을 지원할 예정이다. 그리고 읽기나 쓰기 요구를 함께 모아서 한번에 처리할 수 있는 방법에 대한 연구를 수행할 것이다.

7. 참고 논문

[1] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters", Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327

[2] R. B. Ross, "Providing Parallel I/O on Linux Clusters" ,Second Annual Linux Storage Management Workshop, Miami, FL, October 2000.

[3]M.Vilayannur,M.Kandemir, A.Sivasubramaniam, "Kernel-Level Caching for Optimizing I/O by Exploiting Inter-Application Data Sharing", IEEE International Conference on Cluster Computing (CLUSTER'02),September 2002

[4] Dahlin, M., Wang, R., Anderson, T., and Patterson, D. 1994. "Cooperative Caching: Using remote client memory to improve file system performance", In Proceedings of the First USENIX Symposium on Operating Systems Design and Implemntation. USENIX Assoc., Berkeley, CA, 267-280

[5] Feeley, M. J., Morgan, W. E., Pighin, F. H., Karlin, A. R., and Levy, H. M. 1995. "Implementing global memory management in a workstation cluster", In Proceedings of the 15th symposium on Operating System Principles(SOSP). ACM Press, New york, NY, 201-212

[6] Prasenjit Sarkar , John Hartman, "Efficient cooperative caching using hints", Proceedings of the second USENIX symposium on Operating systems design and implementation, p.35-46, October 29-November 01, 1996, Seattle, Washington, United States

[7] "Linux Kernel Threads in Device Drivers", http://www.scs.ch/~frey/linux/kernelthreads.html