PVFS를 위한 상호 협력 캐쉬의 설계 및 구현

황인철(In-Chul Hwang) 김호중(Hojoong Kim)

정한조(Hanjo Jung) 김동환(Dong-Hwan Kim)

김호진(Hojin Ghim)

한국과학기술원 전자전산학과 전산학전공 컴퓨터구조연구실

2003년 07월 12일

요약

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 파일 캐쉬의 설계와 구현, 그리고 더 높은 성능을 얻기위한 상호협력 캐쉬를 설계하고 구현한다. 그리고 기존 PVFS와의 성능 비교를 통하여 캐쉬의 효율성을 증명한다. 성능 평가 결과 읽기의 경우 PVFS를 위한 상호협력 캐쉬를 사용하는 것이 기존 PVFS에 비하여 더 좋은 성능을 보였다. 그리고 쓰기의 경우 큰 부하 없이 기존 PVFS와 유사한 성능을 나타내었다.

목차

1. 서론

2. 관련 연구

 2.1 PVFS

 2.2 상호 협력 캐쉬

3. PVFS에서 캐쉬 관리자의 설계 및 구현

 3.1 캐쉬 관리자의 설계

 3.2 캐쉬 관리자의 구현

4. PVFS를 위한 상호 협력 캐쉬의 설계와 구현

 4.1 PVFS를 위한 상호 협력 캐쉬의 설계

 4.2 PVFS를 위한 상호 협력 캐쉬 관리자의 구현

5. 성능 평가

 5.1 읽기 수행 결과 및 결과 분석

 5.2 쓰기 수행 결과 및 결과 분석

6. 결론 및 향후 과제

7. 참고 문헌

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 연구 중 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 빠르기 때문에 이런 클라이언트간의 상호협력을 통한 캐쉬를 구성하고자 하는 방식이다. 따라서 상호협력 캐쉬에서는 파일 계층이 자신의 파일 캐쉬, 다른 클라이언트의 캐쉬, 서버 캐쉬, 서버 디스크로 이루어진다. 상호협력 캐쉬에서는 자신의 캐쉬에 파일의 내용이 없을 경우 그 내용을 가지고 있는 다른 클라이언트에게 그 내용을 전달받아서 요구를 처리하는 방법을 사용한다.

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. 그리고 사용자가 임의의 데이터 분산을 할 수 있는 기능을 제공하며 응용프로그램 수정없이

병렬 I/O를 제공받을 수 있다는 장점이 있다. 하지만 PVFS에서는 파일 캐쉬를 제공하지 않으며 단순히 사용자 프로그램에게 파일의 내용을 I/O 서버로부터 전달해 주는 서비스만을 제공한다. 따라서 본 논문에서는 높은 성능을 얻기 위하여 PVFS 파일 시스템 캐쉬와 상호협력 캐쉬를 설계하고 구현한다. 기존 분산 파일 시스템과 달리 병렬 I/O를 제공하는 PVFS의 구조는 메타데이터 관리자와 I/O 서버가 나누어져 있기 때문에 새로운 구조의 상호협력 캐쉬를 설계하고 구현하였다. 그리고 구현된 상호협력 캐쉬를 사용한 PVFS와 기존 PVFS를 비교하여 파일 캐쉬의 성능과 상호협력 캐쉬를 평가한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구로서 PVFS와 상호협력 캐쉬에 대하여 설명한다. 3장과 4장에서는 파일 캐쉬 시스템과 상호협력 캐쉬의 설계와 구현에 대하여 설명한다. 5장에서는 구현된 시스템과 기존 PVFS와의 성능을 평가하고 분석한다. 6장에서는 향후 연구 방향과 결론을 맺는다.

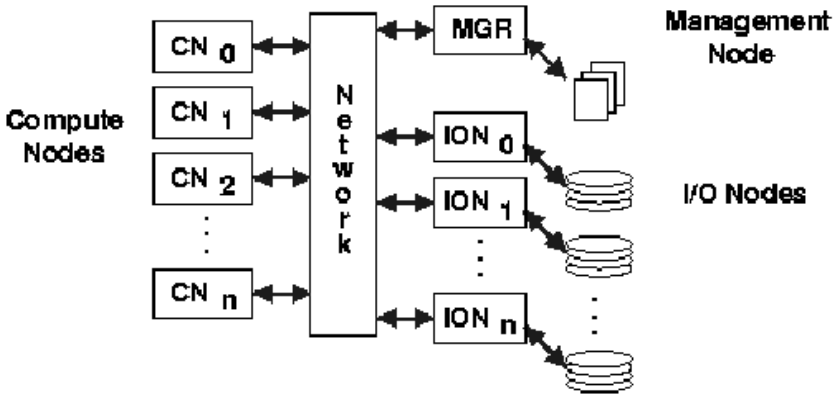
2. 관련연구

본 장에서는 리눅스에서 병렬 I/O를 제공하는 PVFS에 대한 연구들과 분산 파일 시스템에서 성능을 향상시키고자 하는 상호협력 캐쉬에 관한 연구들에 대하여 살펴본다.

2.1 PVFS(Parallel Virtual File System)

PVFS[1,2]는 Linux 운영체제에서 병렬 I/O를 제공하기 위하여 Clemson University에서 구현한 파일 시스템이다. PVFS에서는 기존 응용 프로그램 파일들을 다시 컴파일할 필요없이 그대로 사용하여 병렬 I/O를 제공하여 준다는 특징이 있다. 그리고 사용자가 임의로 데이터 분산을 할 수 있는 기능을 지원한다. 이러한 장점들로 인하여 PVFS는 클러스터 컴퓨팅 환경에서 대용량의 파일 접근을 필요로 하는 응용프로그램에 널리 사용되고 있다.

PVFS의 구조는 다음 [그림 1] 과 같다.

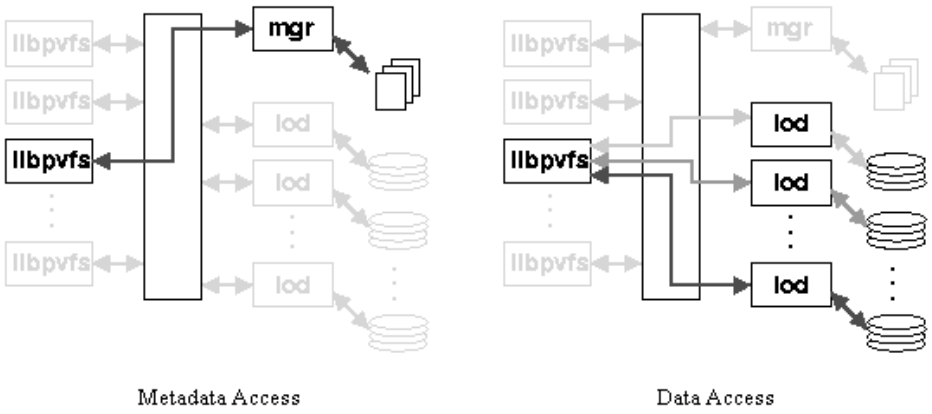


[그림 1] PVFS의 구조

[그림 1]에서와 같이 PVFS는 크게 계산 노드와 관리자 노드, 그리고 I/O 서버로 구성이 된다. 계산 노드는 PVFS의 서비스를 이용하는, 즉 PVFS 파일을 사용하는 사용자이다. 관

리자 노드는 메타데이타 관리자라 부르고, 여기서는 파일에 대한 메타데이타 정보를 지니고 있게 된다. 마지막으로 I/O 서버는 사용자 데이터를 실제 디스크에 저장하는 노드이다.

다음 [그림 2]는 PVFS에서 데이터 접근에 대한 그림이다.



[그림 2] PVFS에서 데이터 접근

[그림 2]에서와 같이 파일을 열 경우에 메타데이터 관리자에게 접근하여 메타데이터 내용을 가져오고 이후의 파일 데이터에 대한 접근은 I/O서버에게 직접 접근하여 이루어 진다. 이러한 방법을 이용함으로서 메타데이터 서버의 부하를 줄인다.

이 중 PVFS의 사용자 접근 방법을 살펴보면 두가지 방법이 존재한다. 첫번째 방법은 사용자 라이브러리를 이용한 방법으로 새로 정의된 라이브러리를 이용하여 프로그램을 재구성하는 방법이다. 이 방법은 응용프로그램을 고쳐야 되는 단점이 있지만 사용자의 접근에 대한 여러 힌트를 가지고서 파일을 효율적으로 접근할 수 있다는 장점이 있다. 다른 한가지 방법은 클라이언트에 PVFS 커널 모듈을 사용하여 기존 UNIX I/O API와 같은 방법으로 접근하는 방법이다. 이 방법은 사용자의 접근에 대한 자세한 정보를 알 수 없다는 단점이 있지만

기존 응용프로그램의 변화 없이 병렬 I/O를 지원받을 수 있다는 장점이 있다.

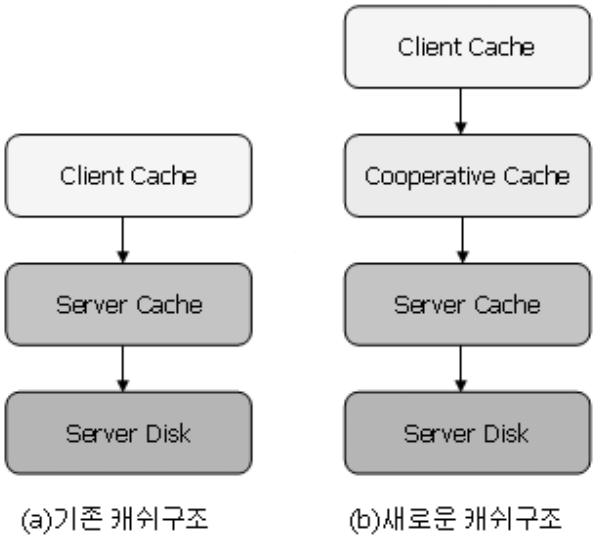
Vilayannur[3]는 기존 PVFS에서 라이브러리를 이용한 방법을 개선하여 PVFS를 위한 단일 노드 캐쉬 시스템을 구현하였다. Vilayannur는 한 노드에서 여러 프로그램이 데이터를 공유할 경우를 가정하여 캐쉬의 효율성을 보여주었다. 그러나 실제 클러스터 환경에서는 여러 사용자가 여러 노드에서 같은 파일에 대한 접근을 하기 때문에 본 논문에서는 Vilayannur와 달리 단일 사용자가아닌 여러 사용자가 커널 모듈을 이용할 경우를 가정하여 상호협력 캐쉬를 커널 모듈을 수정함으로써 구현하였다.

2.2 상호협동 캐쉬

파일 캐슁은 기존 시스템에서 디스크 접근을 줄이고 성능을 높이기 위한 방법으로 널리 사용된다. 클라이언트 캐쉬는 클라이언트의 쓰기/읽기 요구를 처리하는데 사용되고 서버는 클라이언트의 캐쉬에서 처리하지 못한 쓰기/읽기 요구를 처리하는데 사용된다. 이러한 상황에서는 클라이언트가 늘어나면 늘어날수록 서버의 부하가 커지게 되는 단점이 존재하게 되는데 이러한 문제를 해결하기 위하여 클라이언트 사이의 상호협력 캐쉬가 제안되었다 [4].

상호협력 캐쉬는 기존 자료저장 장치의 요구 처리 단계에서 클라이언트가 자신의 요구가 자신의 캐쉬에서 처리가 되지 않을 경우 서버에게 요청하기 전 그 블럭을 캐슁하고 있는 다른 클라이언트에게 그 블럭에 대한 요청을 하여 자신의 요구를 처리하는 방법이다. 이는 서버에게 요청하여 디스크에 접근하는 것보다 다른 클라이언트의 메모리에 접근하는것이 빠르기 때문에 제시되었다. 이에 따라서 기존 클라이언트 캐쉬, 서버 캐쉬, 서버

디스크이던 캐쉬 체계가 다음 [그림 3]에서와 같이 클라이언트 캐쉬, 상호협력 캐쉬, 서버 캐쉬의 구조로 바뀌게 된다.



[그림 3] 상호협력 캐쉬를 사용한 캐쉬 구조

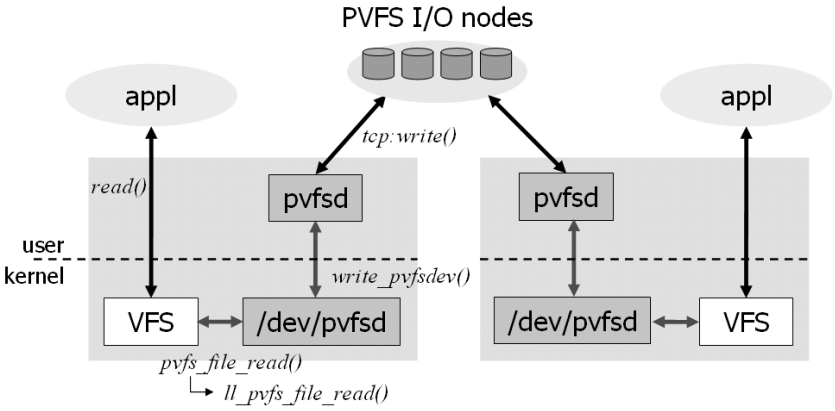
이러한 상호협력 캐쉬를 사용함으로써 서버에게 요청하는 요구의 양이 줄어들게 되어 서버의 부하가 줄어들게 되어 클라이언트의 증가에 대한 확장성이 높아지게 된다. 또한 자신의 캐쉬보다 많은 양의 다른 클라이언트들의 메모리를 서버 디스크에 대한 캐쉬로 사용함으로서 처리할 수 있는 블럭의 수가 늘어나서 성능이 증가하게 된다.

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 캐슁 정보를 가지고 있던 것을 단순한 힌트에 의해 블럭의 캐슁 정보를 유지함으로서 캐슁 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.

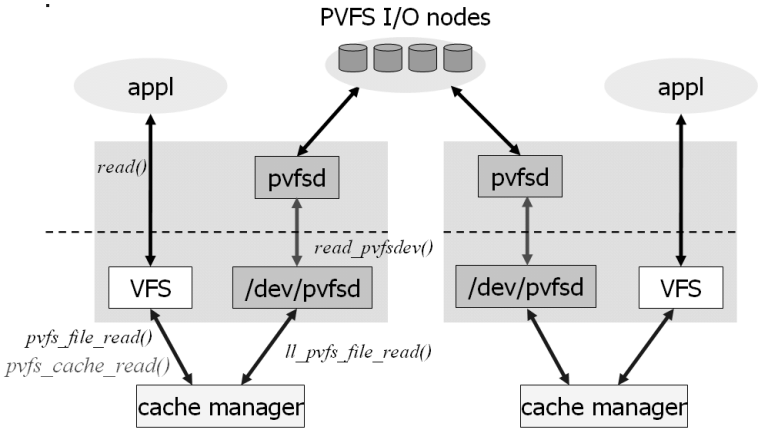
3. PVFS에서 캐쉬 관리자의 설계 및 구현

기존 PVFS에서는 클라이언트 캐쉥을 사용하지 않는다. [그림 4]에서는 기존 PVFS 클라이언트 커널 모듈을 이용한 방법의 구조와 통신을 나타낸다.



[그림 4] 기존 PVFS 클라이언트의 구조와 통신

기존 PVFS에서는 [그림 4]와 같은 구조로 응용프로그램에서 읽기 요구가 가상 파일 시스템으로 전달되고 전달된 내용은 커널 모듈을 통하여 파일 시스템 캐쉬를 사용하지 않고 pvfsd를 통하여 I/O 서버로 전달된다. 이와같은 기존 커널 모듈을 이용한 PVFS에 본 논문에서 제시한 캐쉬 관리자를 추가하여 설계한 모습은 다음 [그림 5]와 같다.



[그림 5] 캐쉬 관리자가 추가된 PVFS 클라이언트의 구조

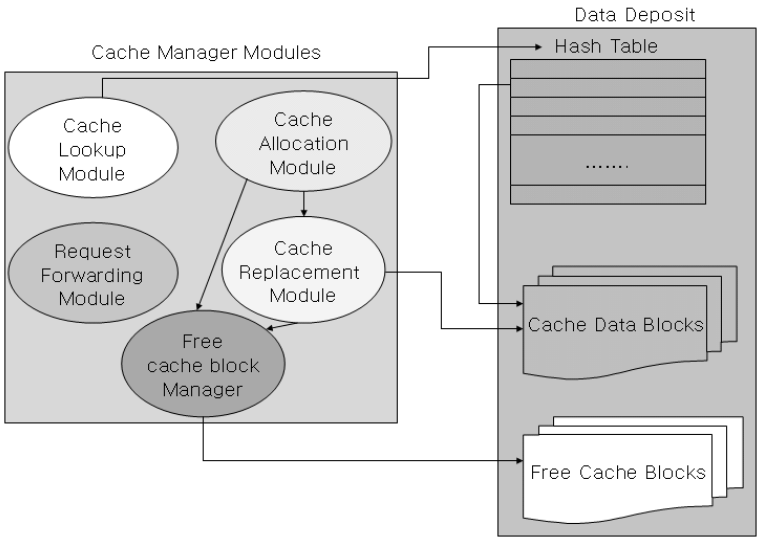
[그림 5]에서와 같이 읽기 요구가 들어왔을때 기존과 달리 캐쉬 관리자에게 그 요구를 캐

쉬 관리자에게 전달하여 자신의 캐쉬에 존재하는 내용은 바로 처리하도록 하고, 자신의 캐쉬에 없을 경우에만 I/O서버에게 넘겨주어 처리하도록 하였다.

그리고 [그림 5]에서와 같이 읽기에 대한 캐쉬를 지원하지만 쓰기에 대한 버퍼링은 지원하지 않고 캐싱된 블럭을 해제시킨 후 기존에 PVFS에서 사용하던 방법을 그대로 사용한다.

3.1 캐쉬 관리자의 설계

캐쉬 관리자의 구조는 다음 [그림 6]과 같다.



[그림 6] 캐쉬 관리자의 구조

[그림 6]에서의 각 모듈은 다음과 같은 동작을 수행한다.

- 캐쉬 찾기 모듈(Cache lookup module) : 요구가 들어오면 그 요구에 맞는 캐쉬 블럭이 있는지 해쉬 테이블을 통하여 찾아본다. 만약 캐쉬 블럭이 존재하면 읽기 요구가 캐쉬 관리자에서 처리되는 것이고, 그 블럭이 없을 경우 요구를 기존 PVFS에서와 같이 요구 전달 모듈에서 I/O 서버에게 요청하여 데이터를 가져와 캐쉬 할당 모듈로부터 새로운 블럭을 할당받아 캐싱한 후 요구를 처리하게 된다.

- 캐쉬 할당 모듈(Cache allocation module) : 새로운 캐쉬 블럭의 할당이나 해제는 캐쉬 할당 모듈에서 관리하게 된다. 먼저 유휴 캐쉬 블럭 관리자에게 비어있는 캐쉬 블럭이 있는지 체크한다. 유휴 블럭이 있을경우 그 블럭을 할당하여 주고 유휴 블럭이 없을 경우에는 유용한 시스템 메모리가 남아 있는지 살펴본다. 시스템 메모리가 많을 경우에는 새로운 블럭을 시스템 메모리로부터 할당받고, 없을 경우에는 캐쉬 대체 모듈로부터 대체된 기존 블럭을 하나 할당받게 된다.
- 캐쉬 대체 모듈(Cache replacement module) : 메모리가 부족해졌을 경우 존재하고 있던 캐쉬 블럭들의 해제는 캐쉬 대체 모듈에서 수행하게 된다. 이 모듈은 일정 시간마다 깨어나서 시스템 메모리를 체크하여 유용한 시스템 메모리의 양이 어느 기준치 이하일 경우 유휴 캐쉬 블럭 관리자에게 요청하여 유휴 캐쉬 블럭을 해제 시키고 그래도 부족할 경우에는 기존 캐쉬 블럭들을 LRU알고리즘에 의해 해제 시킨다.
- 유휴 캐쉬 블럭 관리자(Free cache block manager) : 시스템으로부터 메모리 할당이 되었으나 사용되지 않은 블럭들을 관리하는 역할을 수행한다.
- 요청 전달 모듈(Request Forwarding Module) : 캐쉬 관리자로부터 요청을 받아서 기존 PVFS와 같은 동작으로 요청을 처리하는 역할을 수행한다.

3.2 캐쉬 관리자의 구현

캐쉬 관리자는 3.1절에서 설계된 것을 기반으로 다음 [그림 7]과 같이 구현하였다.

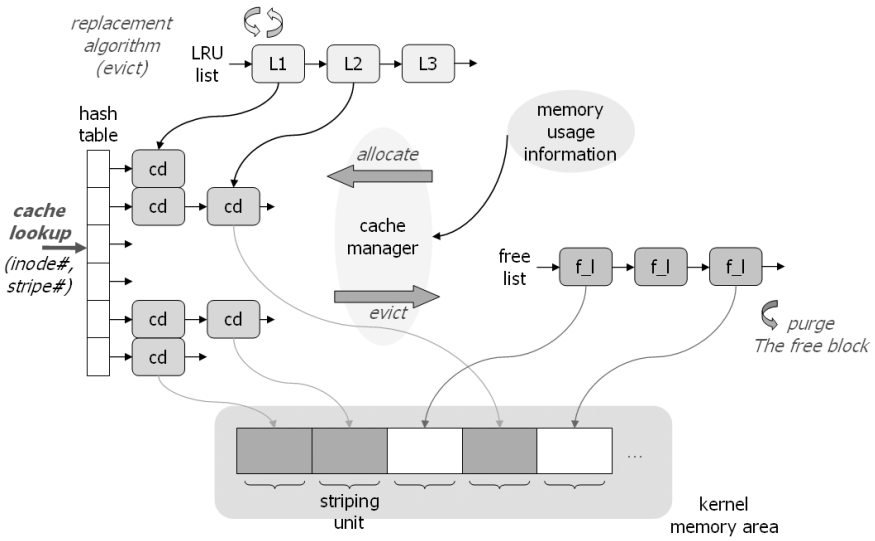


그림 7 캐쉬관리자의 구현

[그림 7]에서 보는 바와 같이 모든 캐쉬 블럭들은 연결 리스트를 사용하여 구현되었고, 대체를 위하여 LRU 리스트를 관리한다. 그리고 할당되었지만 사용되지 않은 블럭들을 유휴 리스트에 관리한다. 캐쉬 블럭은 하나의 분산단위(striping unit)을 기본으로 관리된다. PVFS에서는 분산단위를 기본 64KB로 하며, 시스템에 적합하게 설정이 가능하다.

캐쉉된 블럭이나 LRU 리스트, 유휴 리스트들은 많은 모듈에서 동시 접근 가능하기 때문에 일관성을 위하여 락(lock)을 사용하여 동시성을 추구하였다.

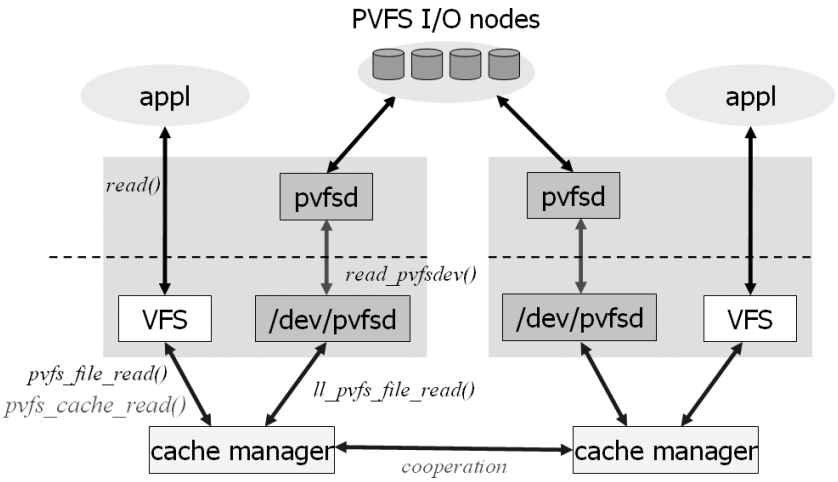
캐쉬를 할당함에 있어 기존 리눅스 커널의 버퍼 캐쉬 시스템을 사용하지 않고 구현하였다. 기존 리눅스 버퍼 캐쉬 시스템을 사용할 경우에는 리눅스 커널을 고치지 않고서는 자신이 가지고 있는 캐쉬 블럭들의 정보를 정확하게 알 수 없기 때문에 이를 사용하지 않는다. 즉, 버퍼 캐쉬 영역을 할당받아 사용할 경우 PVFS 캐쉬 관리자는 시스템에서 임의적으로 캐쉬 블럭을 해제할 경우 자신이 가지고 있지 않는 블럭에 대한 정보의 갱신을 할 수 없기

때문에 리눅스 버퍼 캐쉬를 사용하지 않는다. 따라서 본 논문에서 설계하고 구현한 캐쉬 관리자는 리눅스 캐쉬를 사용하지 않기 때문에 시스템의 메모리 사용량에 따라 캐쉬 블럭을 관리해야 한다. 이를 위하여 캐쉬 대체 모듈에서 시스템 메모리 양에 따라서 알맞게 캐쉬 블럭들을 관리하여 주는 부분을 추가하였다. 캐쉬 대체 모듈은 커널 쓰레드[7]로 구현되어 일정한 시간마다 깨어나서 시스템 메모리를 체크하여 사용가능한 메모리의 양이 일정량 이하로 줄어들면 캐쉬 블럭들을 해제시킨다.

기존 PVFS와 같은 데이터 일관성을 유지하기 위해서 쓰기 요구일 경우 캐슁되어 있는 데이터 블럭을 해제시키는 방법을 이용한다. 따라서 쓰기 요구가 요청되면 기존 PVFS와 같이 I/O 서버에게 내용을 바로 전달받고 씀으로써 데이터 일관성을 유지한다.

4. PVFS를 위한 상호협력 캐쉬의 설계와 구현

기존 PVFS에 상호협력 캐쉬가 추가된 설계된 구조는 다음 [그림 8] 같다.



[그림 8] PVFS를 위한 상호협력 캐쉬

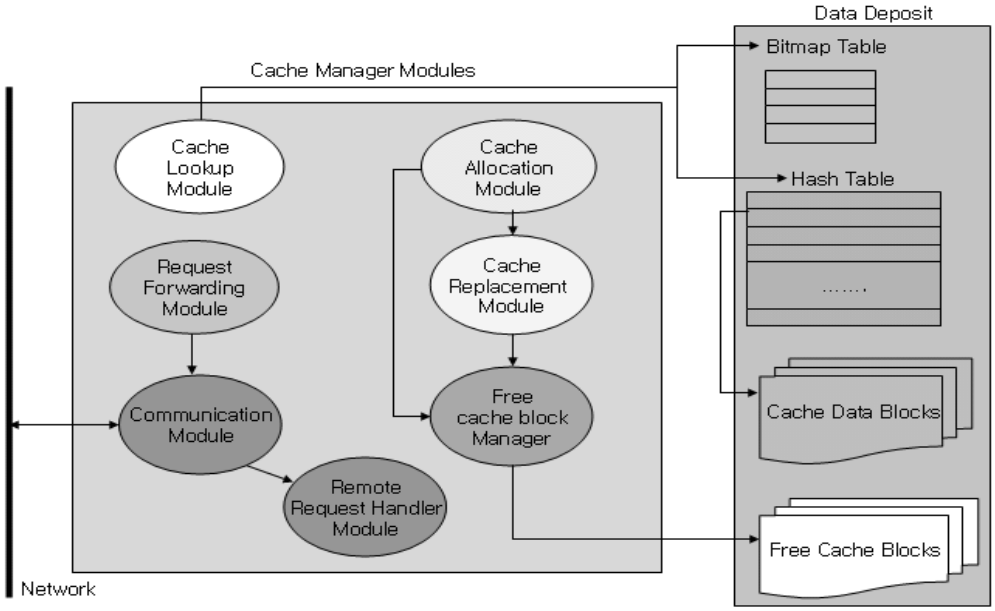
[그림 8]에서와 같이 읽기 요구가 있을 경우 기존 캐쉬 관리자에서 서로 협력을 통하여 읽기 요구를 처리하게 된다. 즉, 읽기 요구가 요청되었을 때 자신의 캐쉬를 먼저 살펴보고 자신의 캐쉬에 데이터가 없을 경우 다른 클라이언트의 캐쉬 관리자에게 데이터를 요청하여 처리한다.

이러한 동작을 하기 위해서는 캐쉬 관리자가 다른 클라이언트가 캐싱하고 있는 데이터의 정보를 가지고 있어야 한다. 기존 연구들에서는 정확한 정보를 갖는 방법[4,5]과 힌트를 이용한 방법[6]이 제시되었다. 힌트를 이용한 상호협력 캐쉬는 정확한 캐쉬 블럭 정보를 가지고 있는 상호협력 캐쉬보다 관리 비용이 싸고 확장성이 뛰어나다는 장점이 있다. 따라서 본 논문에서는 어느정도의 정확성을 지닌 힌트를 기반으로 하는 상호협력 캐쉬를 설계하고 구현하였다. 그리고 병렬 I/O를 지원하는 파일 시스템에 맞지 않는 기존 힌트를

이용한 방법과 달리 본 논문에서 설계하고 구현한 상호협력 캐쉬에서는 병렬 I/O 파일 시스템에 적합한 블럭단위의 일관성을 유지한다.

4.1 PVFS를 위한 상호협력 캐쉬의 설계

다음 [그림 9]는 PVFS를 위한 상호 협력 캐쉬의 구조를 나타낸다.



[그림 9] 상호 협력 캐쉬 관리자의 구조

[그림 9]에서와 같이 기존 캐쉬 관리자에 추가적으로 통신 모듈과 다른 클라이언트로부터 온 요청 처리 모듈이 추가되었다. 그리고 자신이 가지고 있는 캐쉬 블럭들의 정보를 효율적으로 표현하기 위하여 비트맵을 추가하여 설계하였다. 각 모듈의 동작은 다음과 같다.

- 통신 모듈(Communication module) : 통신 모듈에서는 자신의 캐쉬에 있지 않은 데이터를 다른 클라이언트에게 요청하는 메시지를 전달하고 이에 따른 데이터를 전달 받는 동작을 수행한다. 그리고 다른 클라이언트로부터의 데이터 요청을

받아들여 그 요청을 다른 클라이언트로 부터 온 요청 처리 모듈에게 전달한다.

- 다른 클라이언트로부터 온 요청 처리 모듈(Remote request handler module) : 다른 클라이언트로부터 온 요청 처리 모듈에서는 요청받은 데이터를 자신의 캐쉬에 있는지 찾아보고 있으면 전달해 준다. 따라서 본 논문에서 설계한 상호협력 캐쉬에서는 파일 시스템 캐쉬에 여러 데이터 블럭의 복사본이 존재할 수 있다. 그리고 자신이 가지고 있는 파일 블럭에 대한 정보를 다른 클라이언트로 부터 요청받을 경우 자신이 가지고 있는 데이터 블럭에 대한 정보를 전달하여 준다.

4.2 PVFS를 위한 상호협력 캐쉬 관리자의 구현

PVFS를 위한 상호협력 캐쉬 관리자는 캐쉬 관리자와 유사하게 모든 데이터는 연결 리스트로 구현되었다. 비트맵 테이블도 해쉬 테이블과 유사하게 구현되었다.

힌트를 이용한 캐쉬 블럭 정보를 유지하기 위하여 다음과 같은 방법을 이용한다. 기존 PVFS에서는 사용자가 처음 파일을 열 경우 메타데이터 관리자로부터 자신이 열 파일의 메타데이터를 가져와야한다. 따라서 이를 이용하여 메타데이터 관리자에서 기존에 그 파일을 열었던 클라이언트들의 정보를 유지한다. 새로운 클라이언트가 파일을 열 때 메타데이터와 함께 기존 파일을 열었던 클라이언트들의 정보를 가지고 온다. 자신의 캐쉬에 없는 블럭을 읽으려고 할 경우 다른 클라이언트에게 데이터 블럭에 대한 정보 요청과 함께 자신이 가지고 있는 데이터 블럭의 정보를 보내고, 그 요청을 받은 클라이언트는 데이터 블럭에 대한 정보인 비트맵을 보내주게 된다. 따라서 파일 데이터 블럭에 대한 정확한

정보를 유지하지는 않지만 파일 데이터를 많이 사용할 경우 서로 자신의 데이터 블럭 정보를 주고 받게 되어 상호협력을 할 수 있게 된다. 파일이 닫혔을 경우 상호협력 캐쉬에는 그 파일에 대한 캐쉬 블럭들이 존재할 수 있기 때문에 아무런 추가적인 동작도 하지 않고 기존 PVFS와 같은 동작을 수행하게 된다.

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

다음 [그림 10]은 클라이언트에서의 각 동작에 따른 서버들과 다른 클라이언트에게 전달되는 메세지를 나타낸다.

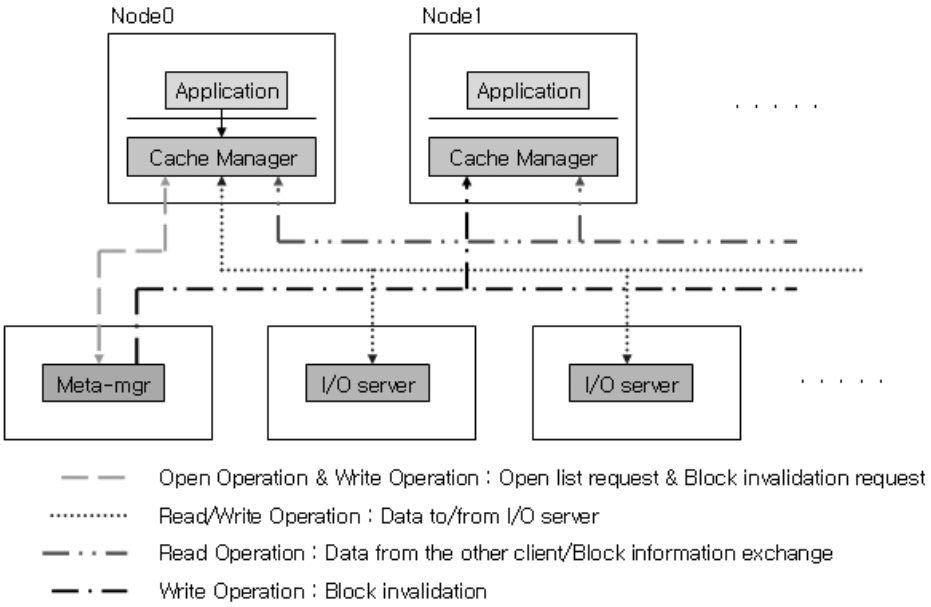


그림 10 각 동작에 따른 메세지의 전달

[그림 10]에서와 같이 처음 파일을 열때 메타데이터 관리자로부터 메타데이터와 이전에 파일을 열었던 클라이언트 리스트를 전달 받는다. 읽기 요구일 경우 자신의 캐쉬나 다른 클라이언트의 캐쉬, I/O 서버에게 요청하여 요구를 처리하고, 다른 클라이언트와 블럭 정보를 교환한다. 쓰기 요구일 경우에는 메타데이터 관리자에게 블럭 해제 요구를 전달하여 메타데이터 관리자가 이전에 파일을 열었던 클라이언트들에게 블럭 해제 메세지를 전달한 후 클라이언트는 자신이 쓴 데이터를 기존 PVFS와 유사하게 바로 I/O 서버에게 전달하여 요구를 처리한다.

클러스터 환경에서는 믿을 수 있는 빠른 네트웍 환경이기 때문에 UDP를 사용할 수 있다. 따라서 PVFS를 위한 상호협력 캐쉬 관리자에서 다른 클라이언트 상호협력 캐쉬 관리자와의 통신을 UDP를 사용하여 구현하였다.

5. 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

한 노드에 메타데이터 관리자를 할당하고 다른 하나의 노드에 I/O 서버를 할당하였다. 그리고 두 노드를 클라이언트로 하여 다음 프로그램을 수행하였다.

- 읽기/쓰기 프로그램 : 읽기/쓰기 요구 크기를 변화시켜 가면서 읽기/쓰기를 수행한다. 읽기일 경우 서버에 캐슁이 되어 있는 경우/없는 경우, 다른 노드에 캐슁이 되어 있는 경우/없는 경우에 따라 여러 번 수행하여 평균값을 계산하였다. 쓰기의 경우에는 캐슁하고 있는 다른 클라이언트가 있을 경우/없을 경우에 따라 여러 번 수행하여 평균값을 계산하였다.

5.1 읽기 수행 결과 및 결과 분석

다음 [표 2]의 식을 이용하여 각 경우에 대하여 읽기 수행 시간을 예측하는 식은 다음과 같다.

Block_Size	캐쉬 블럭 크기. PVFS에서 사용하는 분산 단위와 같다. 본 논문에서는 64KB를 사용하였다.
Req_Size	요청된 쓰기/읽기 크기. 본 실험에서는 64KB에서 4MB까지 사용하였다.
Block_Num	요청된 쓰기/읽기 블럭의 개수. Req_Size/Block_Size와 같다.
Find_Over(B)	Block B를 상호협력 캐쉬 관리자에서 찾는데 걸리는 시간.
Ker_Comm_Over(B)	Block B를 상호협력 캐쉬 관리자끼리 전달하는데 걸리는 시간.
PVFS_Comm_Over(B)	Block B를 기존 PVFS에서 I/O서버 메모리에서 클라이언트까지 전달하는데 걸리는 시간.
I/O_Read_Time(B)	Block B를 디스크로 부터 읽어오는 시간

[표 2] 각 수식들

- 자신의 캐쉬에 읽기 요청된 블럭이 있을때 응답 시간([그림 11]의 cache_hot_coop)

$$Latency_{read\_cache\_hot\_coop} = Block\_Num * Find\_Over(B)$$

- 상호 협력 캐쉬에 읽기 요청된 블럭이 있을때 응답 시간([그림 11]의 coop_hot_coop)

$$Latency_{read\_coop\_hot\_coop} = Block\_Num * (Find\_Over(B) + Ker\_Comm\_Over(B))$$

- I/O 서버의 메모리에 요청된 블럭이 있을때 응답 시간([그림 11]의 iod_hot_coop)

$$Latency_{read\_iod\_hot\_coop} = Block\_Num * (Find\_Over(B) + PVFS\_Comm\_Over(B))$$

- I/O 디스크에 요청된 블럭이 있을때 응답 시간([그림 11]의 iod_cool_coop)

$$Latency_{read\_iod\_cool\_coop} = Block\_Num * (Find\_Over(B) + PVFS\_Comm\_Over(B) + I/O\_Read\_Time(B))$$

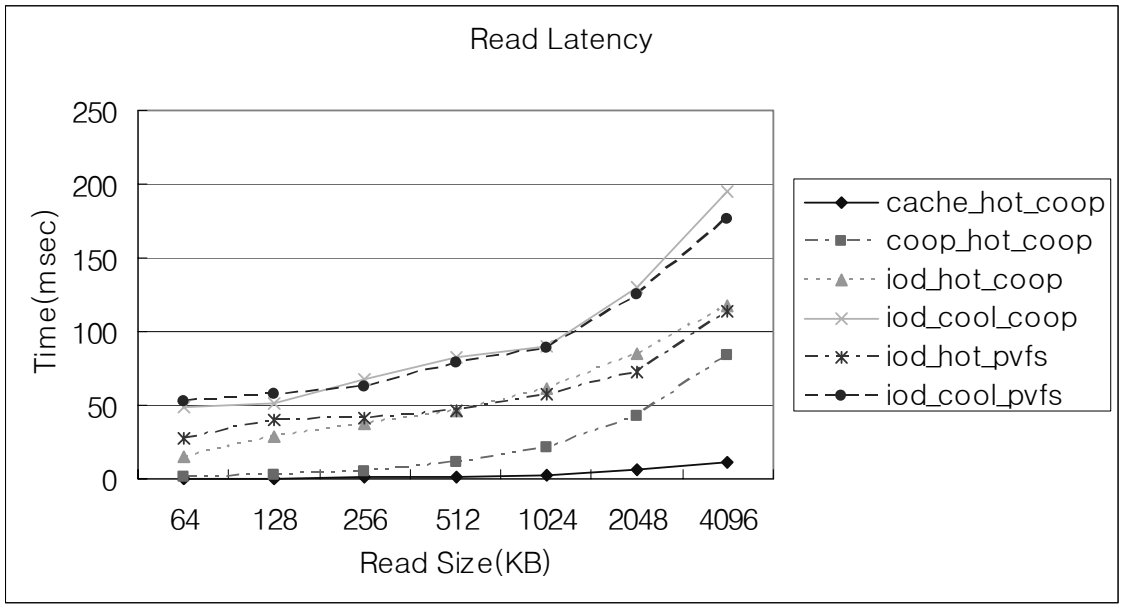
- 기존 PVFS에서 I/O 서버의 캐쉬에 블럭이 있을 때 응답 시간([그림 11]의 iod_hot_pvfs)

$$Latency_{read\_iod\_hot\_pvfs} = Block\_Num * PVFS\_Comm\_Over(B)$$

- 기존 PVFS에서 I/O 디스크에 요청된 블럭이 있을 때 응답 시간([그림 11]의 iod_cool_pvfs)

$$Latency_{read\_iod\_cool\_pvfs} = Block\_Num * (PVFS\_Comm\_Over(B) + I/O\_Read\_Time(B))$$

다음 [그림 11]은 위 여섯가지 경우를 읽기 블럭의 크기를 변화 시켜가면서 읽기를 수행한 수행시간을 나타낸 결과이다.



[그림 11] 읽기 수행 시간

[그림 11]에서 측정한 결과처럼 캐쉬 관리자를 사용한 읽기 성능은 기존 PVFS를 사용하는 것 보다 캐슁을 사용하는 것이 더 좋은 성능을 나타낸다. 그리고 상호협력 캐쉬를 이용하는 것이 I/O 서버에서만 캐슁이 되어 있는 경우보다 더 좋은 성능을 나타내었다. 이는 수식에서 살펴본 바와 같이 캐쉬에서 블럭을 찾고 상호협력 캐쉬 관리자끼리 통신하여

블럭을 전달하는 것이 기존 PVFS에서 I/O서버에 캐슁되어 있는 내용을 전달하는 것보다 빠르기 때문이다. I/O 서버는 사용자 레벨에서 구현되어 I/O서버에 캐슁이 되어 있더라도 이를 시스템으로 부터 가져와 클라이언트에게 전달하는데 있어 많은 부하가 있다.

5.2 쓰기 수행 결과 및 결과 분석

[표 1]의 수식을 이용하여 각 경우를 수식으로 나타내어 보면 다음과 같다.

- 기존 PVFS에서의 응답 시간([그림 12]의 pvfs) :

$$Latency_{write\_pvfs} = Block\_Num * PVFS\_Comm\_Over(B)$$

- 캐슁한 클라이언트가 없을때의 응답 시간([그림 12]의 no_cli) :

$$Latency_{write\_no\_cli} = Block\_Num * (Find\_Over(B) + PVFS\_Comm\_Over(B))$$

- 캐슁한 클라이언트가 있을때([그림 12]의cli_ex) : B' 은 블럭 해제 요청 메세지

$$Latency_{write\_cli\_ex} = Block\_Num * (Find\_Over(B) + Ker\_Comm\_Over(B') + PVFS\_Comm\_Over(B))$$

각 경우에 대하여 쓰기 크기를 다르게 하면서 수행된 결과는 다음 [그림 12]와 같다.

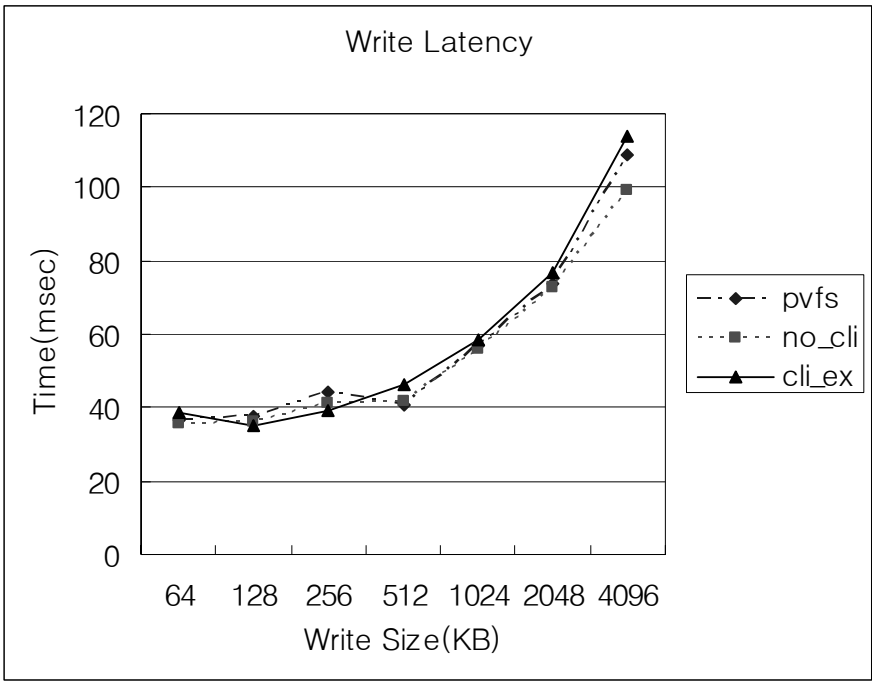


그림 12 쓰기 수행 시간

[그림 12]에서 살펴본 바와 같이 쓰기의 경우는 기존 PVFS에 비해 상호협력 캐쉬 관리자를 사용하는 것이 이득을 볼 수 없다는 것을 알 수 있다. 이런 결과는 본 논문에서 설계하고 구현한 상호협력 캐쉬 관리자에서 쓰기에 대한 어떤 버퍼링도 수행하지 않기 때문이다. 수식에서 살펴보면 기존 PVFS에 비해 상호협력 캐쉬를 사용할 경우 캐쉬 블럭을 찾고 그 블럭을 해제하는 메세지를 보내는 시간이 더 걸리게 된다. 그리고 쓰기는 읽기 요구와 달리 I/O 서버에게 전달된 내용이 I/O 서버에서 모두 버퍼링 되기 때문에 클라이언트에서 I/O 서버 메모리까지 전달되는 시간에 큰 영향을 받는다. 따라서 쓰기 요청일 경우는 기존 방법과 큰 차이 없이 유사한 성능을 나타냄을 알 수 있다. 위 [그림 12]에서 쓰기 블럭의 크기가 128KB에서 256KB일 경우에는 클라이언트가 없는 상호협력 캐쉬 관리자를 사용하는 성능이 더 좋게 나오는데 이 이유는 기존 PVFS를 수행함에 있어 여러 쓰레드들이 어떻게 스케

줄링 되는가에 따라서 빠른 응답을 줄 수 있기 때문에 이와 같은 결과가 나왔다.

6. 결론 및 향후 과제

본 논문에서는 클러스터 컴퓨팅에서 많이 사용되는 Linux 운영체제를 지원하는 병렬 파일 시스템인 PVFS에서 기존에 지원하지 않던 캐쉬 관리자와 효율적인 데이터 관리를 위한 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. 그리고 상호협력 캐쉬를 이용하는 것이 기존 PVFS보다 더 좋은 성능을 보임을 알 수 있었다. 쓰기 요구의 크기를 변경하면서 여러 환경에서 기존 PVFS와 비교하여 보았을 때 큰 차이 없이 유사한 성능을 나타내었다.

본 논문에서는 읽기/쓰기 요구에 대하여 성능평가를 하였다. 성능 평가에서 살펴본 바와 상호협력 캐쉬를 사용하는 PVFS를 사용할 경우 대용량 파일을 접근하는 여러 응용프로그램들의 성능이 개선되고 클러스터 전체의 성능 향상에 큰 도움이 됨을 알 수 있었다. 향후에는 실제 클러스터 시스템에서 동작하는 응용프로그램을 가지고 성능평가와 성능분석을 할 예정이다. 그리고 쓰기의 경우 현재는 버퍼링을 하지 않고 캐싱되어있는 모든 블럭을 해제하고 있으나 쓰기의 경우에서 높은 성능을 얻기 위해서는 버퍼링을 지원할 계획이다. 또한 읽기나 쓰기 요구에 따라 매번 각 클라이언트들에게 블럭을 요청하기보다 그 요청을 함께 모아서 한번에 처리할 수 있는 연구를 수행할 것이다.

7. 참고 문헌

[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