SAN 환경에서의 공유파일 시스템을 위한 광역 버퍼관리기의 설계 및 구현

이경록[0] 김은경 정병수
경희대학교 컴퓨터공학과
{rok000,jlikek}@jupiter.khu.ac.kr jeong@khu.ac.kr

Design and Implementation of Global Buffer Manager for SAN Shared File System

Kyoung-Rok Lee[0] Eun-Kyung Kim Byeong-Soo Jeong
Dept. of Computer Science, Kyunghee University

요 약

최근에는 ATM, Fast Switched LAN, Fiber Channel과 같은 고속의 네트워크의 발달로 인해 분산 환경의 네트워크 파일 시스템에서 디스크를 접근하는 속도보다 원격지 클라이언트의 메모리를 접근하는 속도가 현저하게 증가되었다. 실제로 이와 같은 고속 네트워크 환경을 기반으로 하여 각 서버와 저장 장치를 분리하여 대용량 데이터를 관리하는 SAN(Storage Area Network)과 같은 새로운 네트워크 저장 시스템이 출연하고 있다.

본 논문에서는 이와 같은 새로운 분산 네트워크 파일 저장 시스템 환경에서 필수적으로 고려되어야 하는 광역 버퍼관리기를 설계 및 구현하였다. 본 논문에서 구현된 광역 버퍼 관리기는 크게 데이터 룩업과 버퍼리스트 관리 부분으로 나누어 구성되어 있으며, 이를 위한 적절한 자료 구조와 시스템 내에 있는 각 호스트간의 버퍼블록정보 유지를 위한 방안 및 기존 운영체제의 커널내 버퍼 관리기와 통합하는 방안을 제시한다.

1. 서 론

최근에 전자상거래, 인터넷, 디지털 방송과 같은 새로운 컴퓨팅 환경이 출연함에 따라 폭발적인 대용량 데이터 관리가 요구되고, 또한 값비싼 저장 장치(Storage Device)를 효율적으로 여러 서버에서 공유하여 사용할 수 있는 기술이 필요하게 되었다. 이러한 기술은 기존의 전통적인 네트워크 구조로는 지원할 수 없기 때문에 SAN(Storage Area Network) 구조[1]와 같이 새로운 분산 환경의 고속 네트워크 저장 시스템이 고려되고 있다. SAN에서는 호스트 컴퓨터에서 SCSI를 통하여 스토리지 서버와 신속하게 데이터를 주고받을 수 있는 것처럼 네트워크 상에서 Fiber Channel(FC)의 이점인 고속 전송과 장거리 연결 및 멀티 프로토콜 기능을 활용한다.

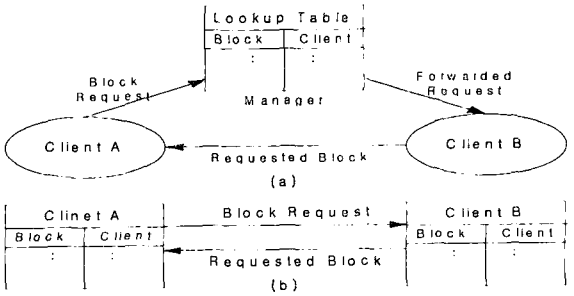
이와 같은 환경에서는 디스크 접근 속도보다 인접 클라이언트의 메모리에 접근하여 자원을 가져오는 속도가 훨씬 더 빠르다. 실제로 인접 클라이언트의 메모리를 활용하여 자원을 공유함으로써 네트워크 파일 시스템의 성능을 개선하는 협력 캐싱(Cooperative Caching) 기술에 대한 연구가 관심을 모으고 진행이 되어 왔다.

이러한 시스템 환경의 변화에 따라 본 논문에서는 고속 네트워크 환경을 이용하여 다중 워크스테이션간의 상호 연결을 지원하는 SAN 저장 시스템 환경에서의 광역 버퍼관리기를 설계하여 속도와 비용 면에 있어서 보다 효율적인 자원의 공유 작업이 수행되도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 살펴보고 3장에 광역 버퍼관리기의 시스템 환경과 설계사항을 제시하며, 마지막으로 4장에는 결론 및 향후 과제를 기술한다.

2. 관련 연구

협력 캐싱을 사용하는 환경에서는 요구한 데이터 블록의 위치에 따라 로컬 캐시 탐색, 원격 캐시 탐색, 디스크 탐색의 세 단계로 이루어진다.



[그림 1] 블록 룩업

원격 캐시의 경우에는 다른 클라이언트 캐시의 블록 정보를 유지하기 위해 [그림 1]과 같은 방법을 사용한다. [그림 1](a)는 관리자를 통해 블록의 정보를 구하여 해당 블록을 얻는 방법이다. 이 경우 관리자의 구성은 클라이언트 전체를 관리하는 관리자를 두는 방법과 클라이언트들을 그룹화하여 그룹 단위당 하나의 관리자를 두는 방법으로 나누어진다. [그림 1](b)의 경우는 각 클라이언트가 전체 블록에 대한 정보를 가지고 있어서 블록 룩업 시에 관리자를 통하지 않는 방법이다.

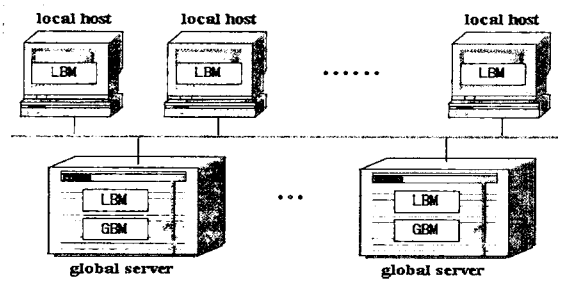
N-chance Forwarding 알고리즘[2]과 GMS 알고리즘[3, 4]에서는 [그림 1](a)와 같은 방법으로 블록 룩업을 한다. 이러한 룩업은 다수의 클라이언트가 동시에 블록을 요구할 때 관리자의 병목현상을 발생시킬 수 있다.

Hint Based 알고리즘[5]의 룩업은 N-chance Forwarding 알고리즘이나 GMS 알고리즘에서와는 달리 [그림 1](b) 방법을 사용한다. 각 클라이언트는 자신이 블록에 대한 힌트 정보를 유지하기 때문에 관리자를 통하지 않고 해당 블록을 가지고 있는 클라이언트에게 직접 요구를 보낼 수 있다.

3. 광역 버퍼관리기의 설계 및 구현

3.1 시스템 환경

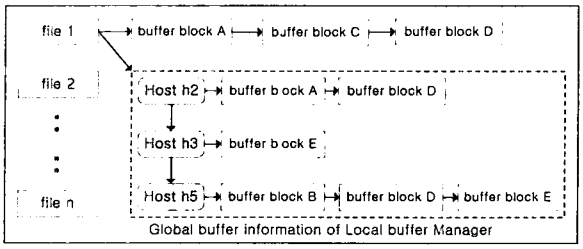
광역 버퍼관리기에서는 다른 지역 서버 메모리에 있는 모든 블록들을 전체적으로 참조할 수 있는 전역적인 자원으로써 다룬다. 분산 네트워크 환경에서 이러한 시스템의 운영을 위하여 관리자를 두는 방법은 관리자에 대한 병목 현상으로 시스템의 성능을 저하시킬 수 있다. 반면에 관리자를 따로 두지 않고 각 클라이언트가 블록에 대한 정보를 가지는 방법은 정보의 부정확성 때문에 불필요한 작업을 발생시킬 수 있다. 따라서 본 시스템에서는 이러한 두 가지 기존 시스템의 문제점을 개선하기 위해 모든 클라이언트들로 하여금 시스템 내의 데이터를 파일 단위로 분산하여 해당하는 관리자가 담당하도록 한다[6]. 그리고 버퍼관리자를 전역 버퍼관리자(GBM)와 지역 버퍼관리자(LBM)의 이원적 운영을 통해서 버퍼 블록을 관리한다. [그림 2]는 전역 버퍼관리자와 지역 버퍼관리자가 어떻게 분산되어 있는지 나타낸다.



[그림 2] GBM과 LBM의 분산 운영

3.2 광역 버퍼관리기의 자료구조

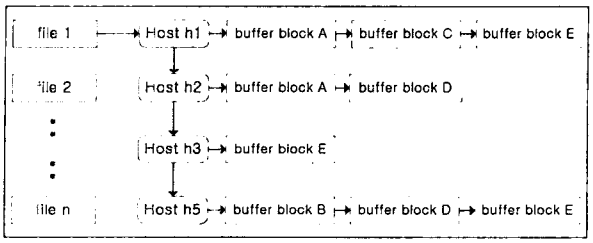
한편 본 시스템에서 각 로컬 호스트는 자신이 관리하고 있는 모든 버퍼 블록을 리스트의 형태로 관리한다. 또한 버퍼 블록을 공유하기 위하여 다른 호스트에서 관리하고 있는 버퍼 블록의 내용을 가지고 있는 전역 버퍼 리스트를 함께 관리하여야 한다.



[그림 3] 로컬 호스트의 버퍼 리스트

[그림 3]은 각 로컬 호스트가 관리하는 로컬 버퍼리스트와 관련 전역 버퍼리스트의 구조를 나타내고 있다.

각 파일은 오브젝트 ID로 식별할 수 있고 파일의 버퍼 블록을 리스트의 형태로 관리한다. 또한 해당 파일의 버퍼 블록을 어느 호스트가 참조하고 있는지, 호스트가 참조하고 있는 블록이 어느 것인지 리스트 형태로 관리한다.



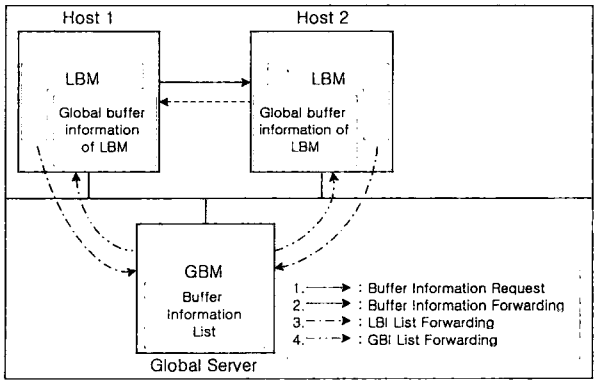
[그림 4] 전역 서버의 버퍼 관리 리스트

[그림 4]는 전역 서버가 관리하는 전역 버퍼리스트의 구조를 나타내고 있다. 전역 서버는 모든 호스트의 버퍼리스트를 통합하여 리스트 형태로 관리하며, 이러한 리스트가 로컬 호스트내의 관련 전역 버퍼리스트에 반영되어 요구하는 블록이 다른 호스트에 있는지 여부를 판단하는데 사용된다.

3.3 룩업 및 버퍼정보 관리

본 시스템에서 임의의 로컬 호스트가 블록을 읽어 들이기 위하여 수행하는 룩업 정책은 요구한 블록의 위치에 따라 크게 지역 메모리 탐색, 원격 메모리 탐색, 디스크 탐색의 세 단계로 이루어진다. 또한 본 시스템에서 모든 호스트는 자신이 참조하는 버퍼 블록의 정보와 함께 다른 호스트의 버퍼정보를 함께 유지하고 있다. 이와 같이 로컬 호스트는 자신의 버퍼 블록의 정보를 공유하기 위해 전역 서버의 전역 버퍼리스트에 자신의 버퍼리스트를 등록하여야 한다. 다음 [그림 5]는 시스템이 자신의 지역 메모리에서 찾지 못한 블록을 얻기 위하여 수행하는

원격 메모리에 대한 룩업 절차와 전역 서버와 로컬 호스트간의 버퍼리스트 정보를 유지하기 위한 절차를 나타낸다.



[그림 5] 룩업 및 버퍼정보 관리

먼저 Host 1이 자신의 로컬 메모리에서 요구한 블록을 찾지 못할 경우에는 LBM의 전역 버퍼 리스트를 참조하여 블록을 갖고 있는 Host 2에 블록 전송을 요구한다. 이에 요청받은 Host 2는 블록 정보를 요청한 호스트에 전송한다. 만일 Host 2에서 블록을 제거하여 블록을 전송하지 못하면 Host 1은 디스크로부터 원하는 블록을 가져와야 한다. 블록을 다른 호스트로부터 전송받았거나 또는 디스크로부터 읽어왔거나 어떤 경우이든지 Host 1은 자신의 로컬 버퍼리스트에 블록에 대한 정보를 추가하고 변경된 버퍼 정보를 전역 서버에 전송하여 전역 버퍼리스트를 최신의 정보로 유지할 수 있도록 한다. 이에 전역서버는 전송받은 로컬 버퍼리스트 정보를 취합하여 로컬 호스트의 관련 전역 버퍼리스트로 전송해주는 기능을 수행한다.

다음 [그림 6]은 Linux kernel의 getblk()함수에 블록 룩업과 버퍼리스트 정보를 관리하는 기능을 추가한 알고리즘이다.

```
sfs_getblk()
{
    ret = getblk();  // Local buffer block search
    if (ret = success)  return;

    else { // LBM내의 GBI에서 block 탐색
                if (GBI에 없으면)
                    ll_rw_block(); // Disk IO 실행
            ......
                // GBI에 해당 block이 있으면
                else {
                    해당 호스트에 buffer block 요청
                    buffer information 갱신
                    ......
                }
    }
}
```

[그림 6] 블록 룩업 및 버퍼관리 알고리즘

이와 같은 본 시스템에서의 블록 룩업 방법은 블록의 위치 정보가 항상 정확하지는 않지만 로컬 호스트가 정확한 블록의 정보를 유지하기 위한 통신 부하와 관리자의 병목 현상을 분산시킬 수 있도록 한다. 한편 전역버퍼정보의 정확성을 유지하기 위하여 로컬 호스트가 버퍼블록을 요구할 때 전역서버와의 사이에 주고받는 메시지와 함께 해당 버퍼리스트를 주고받는다.

4. 결론 및 향후 과제

본 논문에서는 SAN과 같은 고속 네트워크 저장 시스템 환경에서 전역 메모리 버퍼를 관리하기 위한 시스템을 설계 및 구현하였다. 본 논문의 시스템은 크게 데이터 룩업과 버퍼리스트의 관리 부분으로 구성되어 있으며, 이에 필요한 자료 구조 및 시스템 내에서 각 호스트간의 버퍼 블록정보를 유지하기 위한 메커니즘을 포함하고 있다. 실제로 본 시스템의 구현을 위해 Linux Kernel에 필요한 기능을 추가하고 이에 대한 간단한 알고리즘을 제시하였다. 본 논문에서의 시스템은 서버로 선정된 호스트로 하여금 시스템 내의 데이터를 파일 단위로 분산하여 담당하게 하는 분산된 관리자 정책을 사용함으로써 기존 협력 캐싱 기법들에 존재하는 관리자의 병목 현상이나 각 호스트 정보의 정확성 유지에 드는 부하에 대한 문제점을 개선한다.

향후 연구로 본 논문에서 제안된 시스템은 각 호스트의 버퍼 정보 유지를 위해 사용되는 메시지 전송에 있어서 주기적 방법의 사용 및 잠금 관련 메시지의 전송방법에 필요한 통신비용의 측정과 일관성 유지를 위한 연구가 필요하다.

참고 문헌

[1] http://www.san4u.com/san/s1.html

[2] Thomas Anderson, Michael Dahlin, Jeanna Neefe, David Patterson, Drew Roselli, Randolph Wang, "Serverless Network File Systems", In Proceedings of the 15th Symposium on Operating System Principles, 1995.

[3] Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, Henry M. Levy, "Implementing Global Memory Management in a Workstation Cluster", DEC Systems Research Center, 1995.

[4] Michael Joseph Feely, "Global Memory Management for Workstation Networks", A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, University of Washington, 1996.

[5] Prasenjit Sarkar, John Hartman, "Hint based Cooperative Caching", University of Arizona, February 1998.

[6] Toni Cortes, Sergi Girona, Jesus Labarta, "Design issues of a cooperative cache with no coherence problems", Proceedings of the fifth workshop on I/O in parallel and distributed systems, 1997, pages 37-46.