전역적 메모리에서의 블록 룩업과 재배치에 관한 연구

*이 영 섭, 김 은 경, 정 병 수
경희대학교 전자계산공학과
전화 : 031-201-2951 / 핸드폰 : 016-610-7077

A Study on the Block Lookup and Replacement in Global Memory

Young-Seop Lee, Eun-Kyung Kim, Byeong-Soo Jeong
Dept. of Computer Engineering, Kyunghee University
E-mail : akflsqhd@jupiter.kyunghee.ac.kr

Abstract

Due to the emerging of high-speed network, lots of interests of access to remote data have increased. Those interests motivate using of Cooperative Caching that uses remote cache like local cache by sharing other clients' cache. The conventional algorithm like GMS(Global Memory Service) has some disadvantages that occurred bottleneck and decreasing performance because of exchanges of many messages to server or manager. On the other hand, Hint-based algorithm resolves a GMS's server bottleneck as each client has hint information of all blocks. But Hint-based algorithm also causes some problems such as inaccurate information in it, if it has too old hint information.

In this paper, we offer the policy that supplement bottleneck and inaccuracy; by using file identifier that can search for the lookup table and by exchanging oldest block information between each client periodically.

I. 서론

협력 캐싱은 클라이언트 캐시를 조정하고 클라이언트의 지역 캐시에 의해 만족되지 않은 블록 요구를 다른 클라이언트 캐시에 의해 만족되게 함으로써 파일시스템의 성능을 개선하는 기술이다. 최근에 빠른 속도로 네트웍이 발전하고 있기 때문에 이와 같은 협력 캐싱 기술이 가능하게 되었다. 실제로 다음 표 1과 같이 지역에서 캐싱 실패가 발생하면 인접한 클라이언트의 캐시를 이용하는 것이 자신의 디스크에 접근하는 속도보다 빠르다[1]. 지역 디스크의 접근속도는 14,800μs이지만 인접한 클라이언트 메모리의 경우는 6,900(1,050) μs이다.

표 1. 인접 클라이언트 액세스 속도 (단위:μs)
(지역 메모리=250, 지역 디스크=14,800)

Remote	Ethernet		155Mbit/s ATM	
	Mem.	Disk	Mem.	Disk
Mem. Copy	250	250	250	250
Net Overhead Data	400 6,250	400 6,250	400 400	400 400
Disk	--	14,800	--	14,800
Total	6,900	21,700	1,050	15,850

1.1 캐시 블록의 형태

클라이언트 캐시의 블록은 다음과 같이 일반적으로 두 가지 형태로 정의한다[1, 3, 4]; ① 클라이언트가 접근하는 지역 블록, ② 다른 클라이언트들에 의해 클라이언트의 메모리에 저장되어지는 전역 블록. 이러한

지역과 전역 블록의 비율은 그림 1과 같이 클라이언트의 활동 수준에 따라 여러 가지 형태로 결정된다. 활동적인 클라이언트 캐시는 자신의 메모리의 대부분을 접근하기 때문에 지역 블럭이 크고, 비활동적인 클라이언트 캐시는 자신의 메모리 접근 회수가 적기 때문에 전역 블럭이 크다.

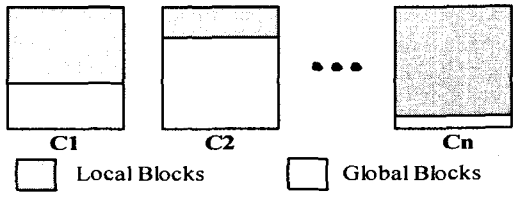

그림 1. 지역 블록과 전역 블록

1.2 블록 룩업(Block Lookup)

협력 캐싱이 사용하는 블록 탐색 정책은 다음과 같다.

① 지역 캐시 search
② 다른 클라이언트의 캐시(remote) search
③ disk search

②의 경우에는 remote 캐시의 블록 정보를 유지하기 위해 그림 2와 같은 방법을 사용한다. 그림 2 (a)는 매니저를 통해 블록의 정보를 구하여 해당 블록을 얻는 방법이다. 매니저의 구성은 클라이언트 전체를 관리하는 매니저[1, 3, 5]를 두는 방법과 클라이언트를 Cluster하여 Cluster 단위당 하나의 매니저를 두는 방법[2]이 있다. 그림 2 (b)의 경우는 각 클라이언트가 블록의 정보를 가지고 있어서 블록 룩업 시에 매니저를 통하지 않는 방법이다[4, 5].

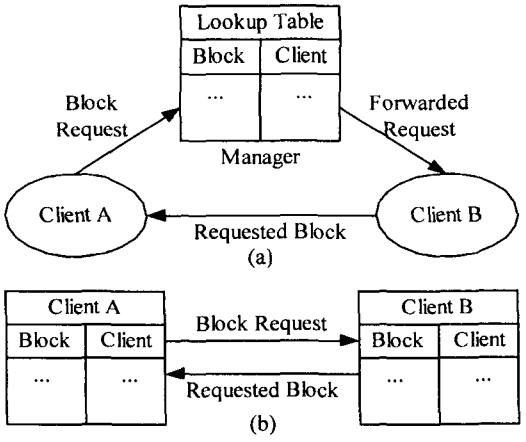

그림 2. 블록 룩업

1.3 블록 재배치(Replacement)

협력 캐싱에서 다른 중요한 캐싱 관리는 재배치이다. 재배치 정책의 목적은 평균 블록 액세스 시간을 최소화하는 것이다. 재배치 정책은 클라이언트의 지역 캐시가 모두 사용되고 있고 새로운 블록이 들어오는 경우에 발생하게 된다. 이때 가치 없는 블록을 제거하고 새로운 블록을 삽입하게 되는데 지역에서는 가장 가치 있지 않은 블록이지만 전체에서는 가치 있을 수 있기 때문에 시스템에서 캐시를 전역적으로 관리해야 한다. 클라이언트 캐시에 있는 블록을 재배치할 때에 할 수 있는 선택은 크게 두 가지가 있다. 만일 블록이 모든 클라이언트에서 가치가 없다면 제거시키고, 그렇지 않으면 다른 클라이언트로 보내 전역 블록을 만들 수 있다[1, 2, 5, 6].

본 논문은 2장에서 협력 캐싱의 두 가지 알고리즘을 소개하고, 본 논문에서 제안하는 전역적 메모리 설계에 대한 방향을 3장에 제시하며, 마지막으로 4장에 결론과 향후 과제를 기술한다.

II. 관련연구

2.1 GMS 알고리즘(Global Memory Service)[3]

GMS에서는 그림 2 (a)와 같은 방법으로 블록 룩업을 한다. 이러한 룩업은 다수의 클라이언트가 동시에 블록을 요구할 때 매니저의 병목현상을 발생시킬 수 있다. 한편 GMS 재배치 정책에서는 클라이언트 캐시 안에 있는 블록의 나이 정보를 수집하고 분산시키는 것을 매니저가 하도록 하며 GMS는 클라이언트 캐시 내의 전역 블록의 중복을 피한다. 이와 같은 이유로 이 시스템에서는 클라이언트 캐시에서 단지 유일한 블록(singlet)만이 협력 캐시에 전송되어진다. 매니저는 블록에 대한 복사본의 숫자를 기억하고 있으므로 클라이언트의 캐시에서 유일한 블록이 생기면 클라이언트에게 통보한다. GMS에서는 기본적 재배치 정책인 LRU(Least Recently Used)를 사용하기 위해 epoch라는 동적인 시간 간격을 정의한다. epoch 동안 매니저는 재배치의 숫자를 결정한다. 그림 3에 나타난 바와 같이 epoch가 시작되면 클라이언트는 재배치를 조정하는 매니저에게 클라이언트 캐시에 있는 모든 블록의 나이에 대한 요약 정보를 보낸다. 그러면 매니저는 각각의 클라이언트 안에 있는 가장 오래된 블록의 위치를 모든 클라이언트에게 알려주기 위해 이 요약 정보를 평가한다. 그 후에 매니저는 클라이언트 i에 대한 w_i를 결정한다(w_i=클라이언트 i에 존재하는 가장 오래

된 블록의 개수). 마지막으로 매니저는 모든 클라이언트의 w_i를 각 클라이언트에게 분산시켜준다. 그리고 클라이언트가 블록을 재배치시킬 필요가 있을 때에는 전달받은 w_i를 이용하여 클라이언트를 선택한다. 이러한 GMS 알고리즘은 매 epoch마다 매니저가 클라이언트와 정보를 교환하고 w_i를 계산해야 하므로 매니저의 오버헤드가 크다는 단점을 가지고 있다.

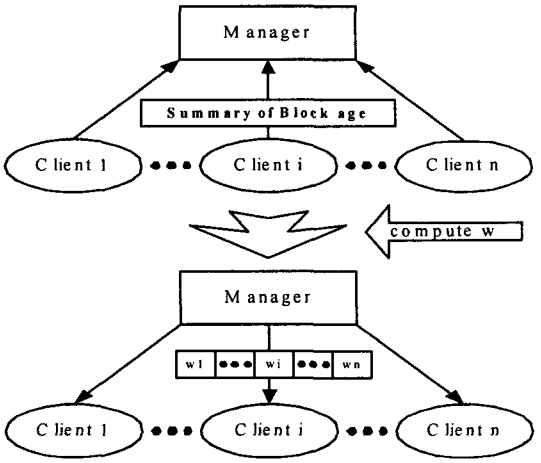

그림 3. GMS에서 epoch

2.2 Hint based 알고리즘[4]

Hint Based에서 룩업은 GMS와 달리 그림 2 (b)를 사용한다. 각 클라이언트는 자신이 블록에 대한 힌트 정보를 유지하기 때문에 매니저를 통하지 않고 해당 블록을 가지고 있는 클라이언트에게 직접 요구를 보낼 수 있다. 알고리즘은 다음과 같다.

① 지역 캐시 실패가 발생했을 때, 클라이언트는 자신이 가지고 있는 힌트 정보를 참조한다.

② 힌트 정보가 블록에 대한 가능한 위치를 가지고 있다면, 클라이언트는 요구를 해당 위치로 보낸다. 그렇지 않다면 서버로 보낸다.

③ 블록에 대한 요구를 받은 클라이언트는 블록이 캐시 안에 존재하는지 검사한다. 존재한다면 블록을 요구한 클라이언트에게 전송한다. 존재하지 않는다면 ②에서부터 다시 시작한다.

힌트 정보는 최초로 클라이언트가 블록을 요구할 때 서버로부터 가져오는데 이러한 힌트 정보가 오래되면 정확성이 떨어지게 된다. 부정확한 힌트 정보를 참조하게 되면 불필요한 작업이 수행되어 시스템 성능이 떨어지게 된다. Hint Base 알고리즘은 블록 재배치 정책을 위하여 그림 4 (a)에서처럼 모든 클라이언트로 하여금 각 클라이언트에 대한 인덱스와 Oldest Block list를 유지하도록 한다. 실제로 그림 4 (b)는 Client A가 자신의 Oldest Block list를 참조하여 Client C를 선택하고 재배치할 블록을 Client C로 보내는 것을 나타낸다. 그런 후에 그림 4 (c)에서와 같이 Client A와 C가 서로 가장 오래된 블록의 나이를 교환함으로써 Oldest Block list를 업데이트하게 된다.

이와 같이 Hint Based 알고리즘의 재배치는 매니저를 통하지 않고 바로 클라이언트간에 이루어지므로 매니저의 병목현상을 줄일 수 있다. 하지만 Oldest Block list가 부정확하면 시스템 성능은 떨어지게 된다.

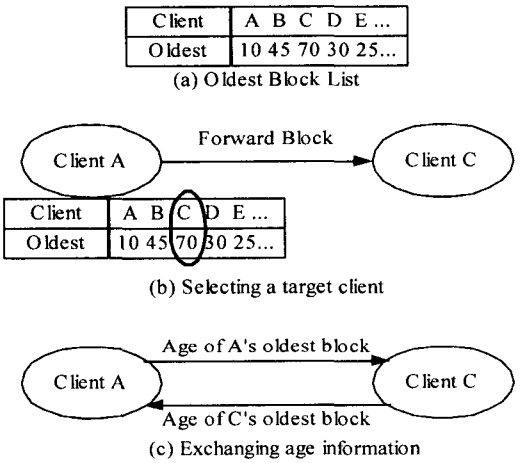

그림 4. Hint Based 알고리즘의 재배치

III. 전역적 메모리 관리자의 설계 방향

3.1 블록 룩업

기존 연구에서 매니저를 두는 방법(그림 2 (a))은 매니저에 대한 병목현상으로 시스템의 성능이 저하될 수 있다. 또한 각 클라이언트가 블록에 대한 힌트 정보를 가지는 방법(그림 2 (b))은 정보의 부정확성 때문에 불필요한 작업을 발생시킬 수 있다. 본 논문에서는 먼저 병목현상을 분산시키기 위해 모든 클라이언트가 자신이 담당하는 블록에 대하여 정보 테이블을 유지하도록 한다. 블록 룩업시에 해당 블록에 대한 정보를 가지고 있는 클라이언트를 찾기 위하여 해쉬 함수를 이용하며, 이때 각 클라이언트는 파일 단위로 블록을 관리한다. 그림 5에서 Client 1이 블록 A를 요구하면 이 블록에 해당하는 파일의 식별자를 통해 이를 담당하는 Client 3을 찾아낸다. Client 3에서는 블록 A가 Client 4에 있으므로 블록 A에 대한 요구를 전송하고, Client 4는 블록 4를 Client 1에게 전송한다. 이와 같은 방법

은 블록의 정확한 위치를 알 수 있게 할 뿐만 아니라 매니저의 병목현상을 분산시킬 수 있게 한다.

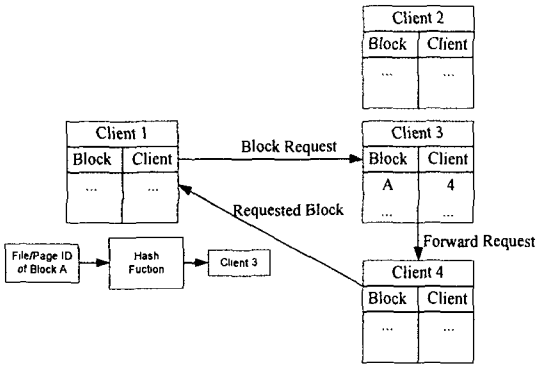

그림 5. 해쉬를 이용한 블록 룩업

3.2 블록 재배치

기존 연구에서는 블록의 재배치를 위해 LRU정책을 사용해 왔지만 이러한 LRU정책을 사용하기 위해서는 모든 클라이언트 블록의 나이정보를 서로 알고 있어야만 한다(전역적인 LRU 정책).

본 논문에서 각 클라이언트는 그림 6과 같은 Oldest 블록의 상태정보를 가지고 있다. 일정한 주기가 되면 각 클라이언트들은 모든 클라이언트들에게 자신의 Oldest 블록의 정보와 상태정보를 전송하게 된다. State는 블록을 저장할 수 있는 공간이 있는지의 여부를 나타내고 있다. 그림 6의 상태정보는 모든 재배치 정책이 기존의 전역적 LRU정책에 의해 실행되는 것을 개선하여 각 클라이언트의 비어있는 메모리 공간을 이용하도록 한다. 본 논문에서 제안한 이와 같은 방법은 각 클라이언트에 있는 Oldest 블록의 개수를 계산함으로써 발생하는 GMS epoch에서의 오버헤드를 줄인다. 또한 Hint based에서의 블록 정보에 대한 부정확성을 크게 개선한다.

Client	A	B	C	...
State	1	1	0	...
Oldest	210	120	130	...

그림 6. Oldest 블록의 상태 정보

IV. 결론

본 논문에서는 다음과 같이 블록 룩업과 블록 재배치의 측면에서 기존 연구의 단점을 개선하였다. 먼저 블록 룩업을 위해서 모든 클라이언트는 자신이 담당하는 블록에 대한 정보를 유지하며, 찾고자 하는 블록에 대한 정보를 가지고 있는 클라이언트를 검색하기 위하여 해쉬를 사용한다. 한편 블록 재배치를 위하여 본 시스템에서는 각 클라이언트가 모든 클라이언트의 Oldest 블록의 상태 정보를 가지고 있으면서 주기적으로 정확한 정보를 교환한다.

이러한 연구를 바탕으로 향후에는 블록 재배치시 주기의 크기 설정 문제와 주기 내에서의 부정확한 블록의 상태 정보 문제, 그리고 전역적 LRU정책을 위한 전역적인 나이 정보 갱신 및 유지 문제를 다양한 환경에서의 실험을 통하여 분석적 결과로서 도출할 것이다.

참고문헌

[1] Michael D.Dahlin, Randolph Y.Wang, Thomas E.Aderson, David A. Patterson, "Cooperative Caching Using Remote Client Memory to Improve File System Performance", Proceedings of the First Symposium on Operating Systems Design and Implementation(OSI 1994).

[2] Toni Cortes, Sergi Girona and 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.

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

[4] Prasenjit Sarkar and John Hartman, "Hint based Cooperative Caching", University of Arizona, 1998.2.

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

[6] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching", In proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 586-595, January 1999. Invited to appear in the special issue of Journal of Algorithms devoted to selected papers form SODA '99