SAN 환경을 위한 효율적인 전역버퍼 관리 알고리즘

IT-based Technology An Efficient Global Buffer Management Algorithm for SAN Environments

이석재*, 박새미**, 송석일***, 유재수****, 이장선*****

충북대학교 정보통신공학과*, LG전자**, 충주대학교 컴퓨터공학과***

충북대학교 전기전자컴퓨터공학부****, (주)매크로임팩트*****

Seok-Jae Lee(sjlee@netdb.cbnu.ac.kr)*

Sae-Mi Park(prettysam@netdb.chungbuk.ac.kr)**, Seok-Il Song(sisong@chungju.ac.kr)***

Jae-Soo Yoo(yjs@cbucc.chungbuk.ac.kr)****, Jang-Sun Lee(sunny@macroimpact.com)*****

요약

분산파일시스템 환경에서는 디스크 접근 비용을 줄이기 위해 각 노드에 캐시된 데이터를 서로 공유하는 협력 캐시 알고리즘이 사용된다. 협력캐시 알고리즘은 분산되어있는 시스템들의 캐시정보를 서로 공유하여 가상으로 더 큰 캐시를 형성함으로써 캐시 히트율을 높이고 디스크 접근을 줄이는 방법이다. 기존에 제안된 협력캐시 알고리즘들은 캐시에 대한 근사정보를 이용하여 메시지 비용을 줄이고, 로컬캐시영역과 글로벌캐시 영역을 가변적으로 사용하여 캐시 히트율을 높이고 있다. 또한 버퍼 교체 시 교체된 버퍼를 비활동적인 노드로 보내어 계속 캐시에 유지하도록 하여 전역 버퍼 히트율을 높이는 장점을 갖는다. 그러나 잘못된 근사정보가 성능을 저하시킬 수 있으며 일관성 유지를 위한 메시지교환 비용이 많이 든다는 단점을 갖고 있다. 또한 비활동적인 노드를 선정하기 위해 사용되는 각 노드의 에이지 정보 관리비용이 많이 드는 단점을 갖고 있다. 본 논문에서는 정확한 캐시정보를 유지하며 일관성 유지비용과 버퍼 에이지 정보 관리비용을 최소화시키는 협력캐시 알고리즘을 제안한다. 그리고 성능평가를 통해 기존의 협력캐시 알고리즘과 비교하여 제안하는 알고리즘의 우수성을 보인다.

□ 중심어 : | 분산 파일 시스템 | 협력캐시 | 교체정책 | 일관성유지 |

Abstract

In distributed file-systems, cooperative caching algorithm which owns the data cached at each node jointly is used to reduce an expense of disk access. Cooperative caching algorithm is the method that increases a cache hit-ratio and decrease a disk access as it holds the cache information of distributed systems in common and makes cache larger virtually. Recently, several cooperative caching algorithms decrease the message costs by using approximate information of the cache and increase the cache hit-ratio by using local and global cache fields dynamically. And they have an advantage that increases the whole field hit-ratio by sending a replaced buffer to the idle node on buffers replacement in order to maintain the replaced cache in the cache field. However the wrong approximate information deteriorates the performance, the consistency maintenance goes to great expense to exchange messages and the cost that manages Age-information of each node to choose the idle node increases. In this thesis, we propose a cooperative cache algorithm that maintains correct cache information, minimizes the maintenance cost for consistency and the management cost for buffer Age-information. Also, we show the superiority of our algorithm through the performance evaluation.

□ keyword : | Distributed File Systems | Cooperative Cache | Replacement Policy | Consistency |

본 연구는 한국과학재단 목적기초연구 (특정기초연구 과제번호 R01-2003-000-10627-0)와 (주)매크로임팩트의 위탁연구과제에 의해서 수행되었습니다.

접수번호 : #031229-002

접수일자 : 2003년 12월 29일

심사완료일 : 2004년 7월 5일

교신저자 : 유재수, e-mail : yjs@cbucc.chungbuk.ac.kr

I. 서 론

분산 파일시스템 환경에서는 데이터를 보다 빠르게 접근해 사용하기 위해서 캐시를 사용한다[6]. 자주 접근되는 데이터는 원격 서버로부터 한 번 읽어와 로컬 사이트의 캐시에 임시 저장하여 사용하므로 캐시된 정보에 대한 반복된 요청은 추가적인 통신비용 없이 로컬에서 처리될 수 있다.

분산 파일시스템 환경에서 사용되는 캐시 알고리즘은 크게 각각의 노드별로 캐시를 독립적으로 관리하는 단일 캐시 알고리즘과 여러 노드가 캐시정보를 공유하여 관리하는 협력캐시 알고리즘으로 나눌 수 있다. 단일캐시 알고리즘은 각 노드가 사용할 수 있는 캐시의 크기가 작기 때문에 캐시 접근 실패가 빈번하고, 따라서 자주 디스크에 접근해야하는 단점이 있다. 또한 접근 빈도가 높은 데이터는 여러 노드의 캐시에 중복해서 저장될 수 있어 캐시 공간의 낭비가 발생하는 단점이 있다. 협력캐시 알고리즘은 분산되어 있는 각 노드들의 캐시 정보를 공유하고 사용함으로써 캐시 접근 실패를 줄여 디스크 접근 비용을 감소시킨다. 또한 전체 시스템 캐시 내의 중복된 데이터 저장을 줄여 공간 활용도를 높이고 있다. 일반적으로 협력캐시 알고리즘이 단일 캐시 알고리즘에 비해 월등히 높은 성능을 보인다[1].

협력캐시 알고리즘은 캐시 정보의 관리 주체에 따라 중앙집중식관리 알고리즘과 분산관리 알고리즘으로 나누어진다. 중앙집중식관리 알고리즘은 모든 노드의 캐시 정보를 중앙의 한 서버가 관리하는 방식으로 관리가 쉽고 구현이 용이한 장점이 있으나, 서버의 과부하로 인해 병목현상이 발생할 수 있고 서버의 오류 발생시 서비스가 불가능한 단점이 있다. 분산관리 알고리즘의 경우 시스템 내의 각 노드가 캐시 정보를 나누어 관리하여 서버의 병목현상이 없고 시스템의 효율성을 높일 수 있는 장점이 있으나, 정보의 일관성 유지를 위한 노드 간 통신비용이 증가하고 관리 방법이 복잡해지는 단점이 있다.

본 논문에서는 분산 관리되는 캐시 정보의 일관성 유지를 위해 사용되는 노드 간 통신비용을 감소시키고 글로벌 캐시의 교체 정책의 복잡성과 관리 비용을 감소시켜 캐시의 효율성을 향상시킬 수 있는 협력캐시 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 기존 분산 파일시스템 환경의 협력캐시 알고리즘들의 특징과 문제점에 대해 기술한다. 3장에서는 본 논문에서 제안하는 협력캐시 알고리즘의 구조에 대해 기술한다. 4장에서는 성능평가를 위한 실험 환경을 기술하며 시뮬레이션 결과와 분석 내용을 기술한다. 마지막으로 5장에서는 결론을 맺고 향후 연구방향을 제시한다.

II. 관련 연구

협력캐시 알고리즘은 [그림 1]과 같이 분산되어있는 시스템들 간에 캐시 정보를 상호협력을 통해 교환함으로써 여러 클라이언트의 메모리들을 하나의 가상적인 전역 캐시로 형성한다[2, 5, 7]. 따라서 글로벌 캐시 적중률이 증가하여 디스크를 좀더 적게 접근하므로 로컬 캐시영역에 데이터를 저장하여 디스크에 자주 접근하는 단일캐시 알고리즘에 비하여 데이터 접근시간이 훨씬 빠르다. 또한 서버가 데이터를 전송하는 대신에 데이터가 있는 클라이언트를 알려주므로 서버에 부하가 적게 걸리며, 비용 면에서 비교할 때 100개의 클라이언트에 값싼 16MB씩 가지는 것은 1개의 서버가 비싼 1600MB를 가진 것보다 훨씬 효율적이다. 가변적으로 이용함으로써 캐시 적중률을 높이는 특징을 가지고 있다.

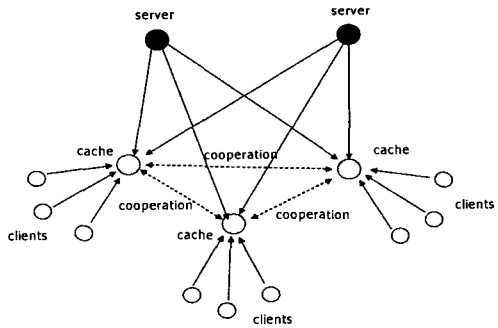


그림 1. 일반적인 협력캐시 알고리즘의 구조

이 장에서는 기존에 제안된 분산 파일시스템에서의 협력캐시 알고리즘인 Hint-based 알고리즘[3], GMS(Global Memory Service) 알고리즘[4]에 대하여 살펴보고 본 논문의 접근 방향에 대해 기술한다.

1. Hint-based 협력캐시 알고리즘

Hint-based 협력캐시 알고리즘은 모든 노드의 캐시 정보를 중앙에서 관리하는 서버 없이 각 노드에 분산 관리하는 알고리즘으로 캐시에 대한 정확한 정보를 이용하기보다 이에 따르는 메시지 통신비용을 줄이기 위한 목적으로 캐시에 대한 근사 정보인 캐시 힌트 정보만을 가지고 처리하는 알고리즘이다.

블록 검색시 클라이언트가 파일 관리자로부터 파일에 대한 토큰을 받으면서 그 파일에 대한 모든 블록의 힌트정보를 함께 받아 그 정보를 이용하여 요청된 블록을 검색하는 방법을 사용한다. 교체정책으로는 항상 서버로부터 가장 처음 복사된 블록(Master copy)만을 전송시키는 방법인 Best-Guess 교체정책을 사용한다. 그리고 각 노드마다 가장 오래된 블록에 관한 정보를 저장하고 있는 리스트를 유지함으로써 교체 시 이 리스트 정보를 이용하여 가장 오래된 블록이 있는 노드에 전송된 블록을 위치시키고 가장 오래된 블록은 버린다. 따라서 Hint-based 협력캐시 알고리즘은 기존 알고리즘에서 발생했던 부가적인 메시지를 효과적으로 줄이고 있으나 각 노드가 블록 정보에 대한 리스트를 유지하고 관리하기 위해 상당한 비용이 든다는 단점을 가지고 있다.

2. GMS 알고리즘

GMS 알고리즘은 Hint-based 협력캐시 알고리즘과 같이 캐시에 대하여 중앙에서 관리하는 서버를 따로 두지 않고 캐시를 분산 관리한다. 또한 캐시를 효율적으로 이용하기 위하여 로컬 캐시영역과 글로벌 캐시영역을 따로 구분하여 사용하지 않고 가변적으로 이용함으로써 캐시 적중률을 높이는 특징을 가지고 있다.

GMS는 교체정책으로 캐시된 페이지에 대해 한번 사용되고 난 후 다시 사용하기까지의 시간간격을 에이지(Age) 정보라고 정의하고 그 정보를 이용하여 전체 캐시에서 가장 자주 사용되는 페이지를 우선적으로 유지하는 정책을 사용한다. 따라서 기존의 알고리즘에 비하여 전체적인 시스템 성능은 향상시키고 있으나, 데이터의 가치판단을 위한 정보관리 및 데이터관리 비용이 상당히 높다는 단점을 가진다.

III. 협력캐시 알고리즘 설계

본 논문에서는 캐시에 대해 중앙에서 관리하는 서버 없이 캐시를 분산 관리하며 전역 캐시적중률과 로컬 캐시 적중률을 동시에 만족시키는 효율적인 캐시 알고리즘을 설계한다. 중앙에서 관리하는 서버 없이 캐시를 분산 관리하게 되면 캐시에 대한 정보를 관리하기 위한 비용이 많이 들고 블록접근시간도 지연되지만, 중앙 서버가 캐시를 관리함으로써 생기게 되는 서버의 병목현상이 발생하지 않는다.

이에 본 논문에서는 서버 없이 캐시를 분산 관리하여 서버의 과부하를 막고 데이터 관리비용을 최소화시키는 방법에 초점을 맞추어 설계한다. 또한 효율적인 글로벌 캐시 적중률과 로컬 캐시 적중률을 위하여 글로벌 캐시영역과 로컬 캐시영역을 따로 구분하여 사용하지 않고 동적으로 유연하게 이용함으로써 캐시 적중률을 높인다. 그리고 글로벌 캐시 적중률을 향상시키기 위하여 한 노드에서 교체된 캐시를 모든 노드 중 가장 비활동적인 노드로 전송하는 방법으로 설계하며, 비활동적인 노드의 선정방법은 전체 알고리즘의 성능에 큰 영향을 주므로 메시지 교환을 최소로 하며 CPU 점유가 적은 노드 선정방법으로 제안한다.

본 장의 구성은 먼저 전체 시스템의 구성도에 대해서 기술하고 논문에서 제시하는 캐시의 메타정보 관리 방법에 대해서 기술한다. 그리고 노드가 블록에 접근하기 위해 블록의 위치정보를 검색하게 되는데 이와 관련된 로컬 히트 처리방법과 페이지 폴트 처리방법에 대해서 기술하고, 캐시 적중률을 최대화하기 위한 캐시의 교체정책에 대해서 기술한다. 마지막으로 임의의 노드에서 발생한 블록접근 요청에 대해서 항상 최신의 데이터를 제공하기 위해 필요한 캐시의 일관성 유지에 대한 설계 내용을 기술한다.

1. 전체 구성도

[그림 2]는 본 논문에서 제안하는 알고리즘의 전체 구성도이다. [그림 2]와 같이 여러 개의 노드가 SAN 스위치에 연결된 저장장치를 공유한다. SAN은 서버에 개별적으로 연결되던 저장 시스템을 광 채널과 같은 고속의 전용 네트워크에 직접 연결하여 중앙 집중적인 저장 시스템의 관리를 가능하게 하고, 서버를 거치지 않고 네트워크에 연결된 저장 장치를 직접 접근할 수 있게 만들어주는 분산파일시스템 환경을 제공하여 좀 더 효과적으로 저장 장치에 저장된 데이터를 접근하여 변경 및 읽기를 수행한다.

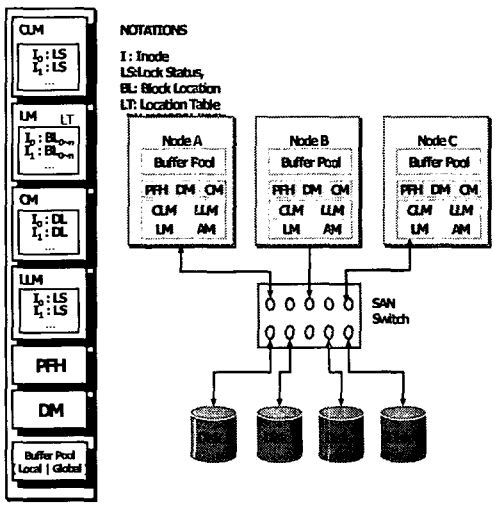


그림 2. 전체 구성도

[그림 2]의 왼쪽부분은 하나의 노드에 대한 구성도로 각 노드는 전체 노드의 잠금(Lock)을 관리하고 있는 CLM(Cluster Lock Manager)과 자신 노드의 잠금을 관리하는 LLM(Local Lock Manager), 그리고 제안하는 알고리즘의 주요소인 LM(Location Manager), CM(Consistency Manager), DM (Delibery Manager), PFH(Page Fault Handler), AM(Age Manager)으로 구성된다.

LM은 각 노드에 위치하여 노드에 할당된 블록들의 위치정보를 관리하는 관리자이고, CM은 캐시의 일관성을 유지하는 일을 수행하는 관리자이다. DM은 노드의 블록을 요청한 노드에게 전송시키는 작업을 수행한다. PFH는 페이지 폴트가 일어났을 때, 즉 노드가 원하는 블록을 로컬 캐시영역에서 검색하는데 실패하게 될 경우 이를 해결하는 모든 절차를 수행해주는 관리자이다. 그리고 AM는 가장 비활동적인 노드를 선택하기 위한 정보를 제공하는 관리자이다. 제안하는 알고리즘에서는 캐시 교체시 가장 비활동적인 노드 즉, 오랫동안 참조되지 않은 블록의 수가 많은 노드를 찾아서 교체하게 될 블록을 전송하는데 이에 대한 정보를 관리한다.

2. 캐시의 메타정보 관리

제안하는 알고리즘에서는 블록의 정보들을 기술하는 메타정보를 관리한다. 메타정보에는 블록의 현재위치, 블록의 변경 여부, 블록의 중복 여부, 블록의 최신본 여부의 정보들이 있다. 그리고 이 메타정보는 LM에 의해서 파일단위로 관리가 되며 LM은 각 파일이 담당하는 블록들의 정보들을 위치 정보 테이블을 이용하여 관리한다. 또한 파일 단위로 라운드 로빈 방식을 통해 각 노드가 정보를 분산 관리한다.

2.1 위치정보 테이블

제안하는 알고리즘에서는 캐시의 메타정보를 위치정보 테이블을 이용하여 관리한다. 위치정보 테이블의 구조는 각 파일에 대한 정보를 기억하기 위해 만들어진 inode 번호와 inode에 따른 블록 번호 그리고 해당 블록의 위치정보들로 구성된다. 블록의 위치정보는 각 노드마다 2비트씩 할당하여 처음 비트는 해당 노드에 블록의 존재 여부를 나타내고, 두 번째 비트는 블록의 최신정보 여부를 나타낸다. [그림 3]은 세 개의 노드에 대한 위치정보 테이블의 예이다.

Inode 번호 / 블록 번호 / 블록 소유 노드 / 최신 블록 여부

Inode Nos	Block IDs	Location					
		N1		N2		N3	
2	12	1	0	1	1	1	1
	13	0	0	0	0	1	1
	14	1	1	1	1	0	0
5	32	1	0	1	1	0	0
	33	0	0	1	1	0	0
	40	1	1	0	0	0	0
	42	1	1	1	0	1	0

그림 3. 블록의 위치정보 테이블

2.2 수정 리스트

쓰기 잠금을 가진 노드가 블록에 새로운 정보를 써서 블록의 메타정보가 수정되어야 할 경우, 제안하는 알고리즘에서는 수정된 정보를 바로 위치정보 테이블에 반영시키지 않고 우선 수정 리스트에 정보를 유지한다. 수정 리스트는 메시지 전송을 줄이기 위한 것이며 수정 리스트의 구조는 간단하게 1바이트로 구성되며 변경된 블록만을 저장한다. 따라서 블록 검색 시 원하는 블록을 찾은 후에 항상 수정 리스트를 참조하여 블록의 상태 정보를 확인한다.

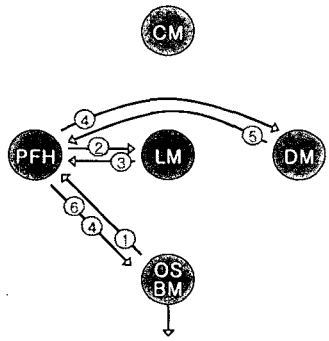
3. 페이지 검색

각 노드는 원하는 블록에 접근하기 위하여 먼저 자신의 로컬 캐시영역을 검색한다. 블록 검색이 로컬 캐시영역에서 성공적으로 완료되면 로컬 히트라고 하고, 블록 검색이 로컬 캐시영역에서 실패하면 페이지 폴트가 발생되었다고 한다.

3.1 페이지 폴트 처리

노드가 원하는 블록을 로컬 캐시영역에서 검색하는데 실패한 경우, 즉 페이지 폴트가 발생하게 되면 PFH는 [그림 4]와 같이 처리한다.

PFH는 우선 블록의 위치를 검색하기 위해 LM에게 질의를 내린다. LM은 블록 위치를 검색하여 응답을 하는데 블록이 다른 노드의 캐시영역에 존재하지 않는 경우에 디스크로부터 블록의 전송을 지시함으로써 페이지 폴트 처리를 수행한다. 하지만 블록이 다른 노드의 캐시영역에 존재하는 경우, 블록의 위치정보를 PFH에게 응답하고 PFH는 다시 블록의 위치정보를 가지고 DM에게 블록 복사본의 전송을 요청한다. 그리고 DM는 블록이 위치한 노드의 캐시영역으로부터 블록 복사본을 전송한다. 이 때 블록이 그 노드의 캐시영역에 있지 않아서 전송하지 못하는 경우가 발생할 수 있다. 이것은 PFH가 블록의 위치정보를 받고나서 블록을 요청하는 사이에 블록의 위치가 교체정책에 의해 변경되었다는 것을 의미한다. 따라서 이럴 경우에 블록을 가져올 수 없으므로 디스크로부터 원하는 블록을 요청함으로써 페이지 폴트처리를 완료한다.



- 1 : page fault 처리 지시
- 2 : 블록 위치 검색
- 3 : 블록 위치 응답 (NodeID or NULL)
- blue 4 : 3의 응답이 NULL, disk I/O 지시
- read 4 : 블록 전송 요청
- 5 : 블록 전송
- 6 : DM에 해당 블록이 없으면 blue 4 와 동일
  그렇지 않으면 블록을 캐시영역에 위치

그림 4. 페이지 폴트 처리

3.2 블록 연산과 블록의 위치정보 유지

[그림 5]는 노드 N1이 블록 14를 읽기 연산을 수행하기 전과 읽기 연산을 수행한 후의 LT를 나타낸다. 먼저 블록이 있는 파일에 대한 잠금을 얻어온 후 위치정보 테이블에서 읽기 연산을 수행시킬 블록의 위치정보를 검색한다. 그리고 검색과 동시에 위치정보 테이블에 요청한 노드가 블록을 가지고 있는 것으로 블록 상태를 변경시킨다.

▫ 블록 요청 수행전의 위치정보 테이블

Inode 번호	블록 번호	블록 소유 노드 / 최신 블록 여부					
Inode Nos	Block IDs	N1		N2		N3	
2	12	1	0	1	1	1	1
	13	0	0	0	0	1	1
	14	0	0	0	0	1	1

▫ 블록 요청 수행후의 위치정보 테이블

Inode 번호	블록 번호	블록 소유 노드 / 최신 블록 여부					
Inode Nos	Block IDs	N1		N2		N3	
2	12	1	0	1	1	1	1
	13	0	0	0	0	1	1
	14	*1*	*1*	0	0	1	1

그림 5. 읽기 연산과 위치정보 테이블 변경

[그림 6]은 노드 N1이 블록 14를 쓰기 연산을 수행하기 전과 쓰기 연산을 수행한 후의 LT를 나타낸다. 우선 위치정보 테이블에서 쓰기 작업을 수행시킬 블록의 위치정보를 검색한다. 그리고 쓰기 작업 후에는 그 블록정보만 변경되므로 요청한 노드만이 최신의 블록을 가지고 있는 것으로 변경시킨다. 또한 수정 리스트에 수정된 블록 ID를 넣어 블록의 변경을 표시한다.

■ 블록 요청 수행전의 위치정보 테이블

Inode 번호	블록 번호	블록 소유 노드 / 최신 블록 여부					
Inode Nos	Block IDs	N1		N2		N3	
2	12	1	0	1	1	1	1
	13	0	0	0	0	1	1
	14	0	0	0	0	1	1

■ 블록 요청 수행후의 위치정보 테이블과 수정리스트

Inode 번호	블록 번호	블록 소유 노드 / 최신 블록 여부					
Inode Nos	Block IDs	N1		N2		N3	
2	12	1	0	1	1	1	1
	13	0	0	0	0	1	1
	14	1	1	0	0	1	0

수정리스트 (dirty list)	16	35	13	..

수정리스트 (dirty list)	16	35	13	14

그림 6. 쓰기 연산과 위치정보 테이블 변경

4. 교체정책

제안하는 알고리즘에서는 캐시 교체시 비활동적인 노드, 즉 오랫동안 참조되지 않은 블록의 수가 많은 노드를 찾아 그 노드에서 가장 오랫동안 사용하지 않은 블록과 교체한다.

4.1 캐시의 에이지 정보

AM는 비활동적인 노드를 선택하기 위해 다음과 같은 에이지 정보를 제공한다. 각 노드는 일단 블록이 처음 사용이 되고 다시 사용되기까지의 경과된 시간을 N단계로 나누는데, 단계가 높을수록 경과시간이 오래됨을 의미한다. 그리고 가장 높은 단계에 해당하는 블록의 개수를 유지하여 다른 노드들과 이에 대한 정보를 서로 교환한다. 이때, 블록 검색이나 블록 전송시 메시지를 교환하게 될 때 에이지 정보를 첨부하여 같이 보냄으로써 부가적인 메시지 수를 줄인다. 그리고 일정 시간동안 메시지를 주고받지 않은 노드에 대해서는 강제적으로 메시지를 전송시킴으로써 에이지 정보를 수집하게 된다. AM은 메시지 교환시 얻은 상대 노드에 대한 에이지 정보를 Age 리스트에 저장하고 캐시 교체시 높은 단계의 에이지에 해당하는 블록의 개수가 많을수록 비활동적인 노드로 간주한다.

4.2 교체 알고리즘

먼저 노드에서 에이지가 가장 많은 블록을 우선순위로 하여 교체할 블록을 선택한다. 그리고 AM에 의해 수집된 Age 리스트를 참조하여 교체할 블록을 전송할 노드를 선택한다. 노드 선택의 기준은 먼저 에이지가 가장 많은 단계를 가지고 있는 노드를 우선순위로 하며, 수정리스트를 이용하여 선택된 노드의 가장 오래된 블록의 유효성을 검증하여 블록을 버리거나 디스크에 쓰고 전송된 블록은 그 블록의 자리로 위치시킴으로써 캐시교체를 완료한다.

5. 잠금 연산

제안하는 알고리즘에서는 원하는 블록에 접근하기 위해서 파일단위의 잠금을 얻어 접근한다. 먼저 노드가 파일에 대한 잠금을 요청하게 되면 전체 노드의 잠금을 관리하고 있는 CLM이 그 파일에 대한 잠금을 가지고 있는 노드로부터 잠금을 얻는다. 그리고 그 파일에 대한 수정 리스트를 그 노드의 CM으로부터 함께 가져온다. 그리고 LM에게 수정리스트에 대한 pruning 작업을 요청한다. prunning 작업은 수정 리스트에서 필요 없는 블록 ID를 제거하는 일로 복사본이 없는 유일한 블록들의 경우 수정 리스트에 저장할 필요가 없으므로 위치정보 테이블을 검색하여 복사본이 있는 블록들만을 수정 리스트에 유지하고 복사본이 없는 블록들은 수정 리스트에서 삭제한다. LM은 pruning 작업을 마친 후 CLM에게 다시 수정 리스트를 전달하고 CLM는 잠금을 요청한 노드에게 잠금 허

용과 동시에 prunning 작업을 마친 수정 리스트도 함께 보낸다.

6. 일관성 유지 방법

제안하는 알고리즘에서는 블록의 내용이 변경될 때마다, 모든 다른 복사본의 내용을 곧바로 변경시키지 않고, 우선 수정 리스트에 변경 정보를 저장한 뒤, 변경된 블록을 접근하게 되었을 때 수정 리스트를 참조하여 변경시키는 방법인 LWI(Lazy Write Invalidate) 알고리즘을 사용한다. LWI 알고리즘은 어느 시점에서 논리적으로 보았을 때 서버가 최신의 파일 자료를 가지지는 못하기 때문에 캐시된 데이터들의 일관성이 유지되지 않은 듯 보일 수 있지만, 변경된 블록을 접근하게 되었을 때 변경된 정보가 반영되기 때문에 전체적으로 데이터의 일관성을 유지하고 있다. 이 방법은 블록의 내용이 변경될 때마다 일일이 다른 복사본들의 내용을 변경시켜주기 위해서 발생되는 추가적인 메시지 교환을 줄일 수 있다는 장점을 가진다.

Ⅳ. 성능 평가

본 장에서는 Ⅲ장에서 설계한 협력캐시 알고리즘과 기존의 협력캐시 알고리즘들을 시뮬레이션을 통해 성능 평가를 수행하고 결과를 분석하여 기술한다.

1. 시뮬레이션 환경

시뮬레이션 구현에는 Window XP 운영 체제에 Intel Pentium(R)4 Xeon 1.80GHz Dual CPU, 2GB의 메인 메모리를 갖는 시스템이 사용되었으며, Visual C++ 6.0을 사용하여 구현하였다. 성능 평가를 위해 약 3000개의 파일에 접근하는 52만 여개의 블록 I/O 요청 값을 데이터로 사용하였다. 실험 데이터는 연구팀이 공유하여 사용 중인 파일 서버에 접근하는 블록 I/O 요청을 72시간 동안 수집하였다. 성능평가에 사용된 파라미터는 [표 1]과 같다.

표 1. 성능 평가를 위한 파라미터들

파라미터	값
I/O Request	524280
Node	8개
Client Cache Size	16 ~ 128MB
Server Cache Size	128MB
Block Size	8KB
Local Memory Access Time	0.25 ms
Remote Memory Access Time	1.25 ms
Disk Access Time	15.85 ms
Write policy	write-through
Forwarding Cache Entries	100
Message Latency	0.2 ms

2. 시뮬레이션 결과

시뮬레이션에서는 제안한 알고리즘을 GMS 알고리즘, Hint-based 알고리즘과 비교하여 성능을 평가하였다. 평가 기준은 캐시 적중률, 블록 접근시간, 에이지 정보 관리비용 그리고 일관성 유지를 위한 메시지 수이다.

2.1 캐시 적중률 비교

제안하는 알고리즘에서는 요청된 페이지에 대해서 로컬 캐시영역과 글로벌 캐시영역에서 페이지를 찾았을 경우 그 비율을 캐시 적중률이라 정의하고 기존 알고리즘들과 비교평가 하였다. 제안하는 알고리즘과 기존의 알고리즘의 캐시 적중률을 비교한 결과 [그림 7]처럼 128MB에 대해서 제안하는 알고리즘과 GMS 알고리즘이 35%로 Hint-based 알고리즘에 비하여 약 2% 높은 캐시 적중률을 보이고 있으나 세 알고리즘이 거의 비슷한 성능을 보이고 있다. 이것은 세 알고리즘 모두 교체정책으로써 캐시에서 가장 오래된 블록을 교체하는 방법을 사용하기 때문이다.

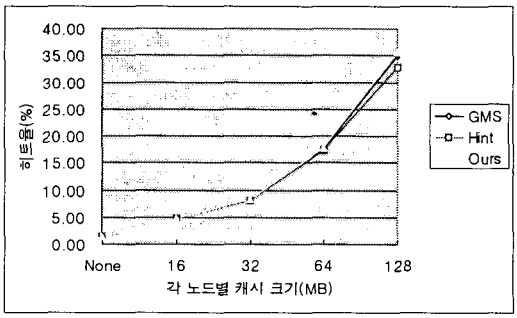


그림 7. 캐시 적중률

2.2 블록 접근시간 비교

제안하는 협력캐시 알고리즘의 성능을 평가하기 위하여 블록에 접근하는 질의를 수행하여 블록의 평균 접근시간 측정한다. [그림 8]은 각 노드의 캐시 크기를 16~128MB로 변경하면서 수행한 평균 블록 접근시간을 나타낸다.

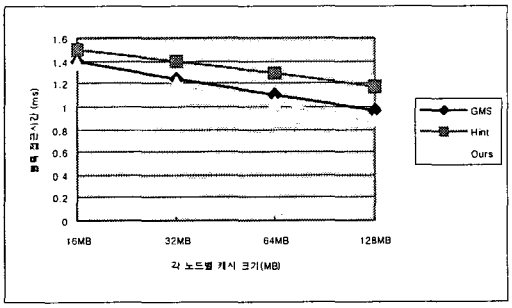


그림 8. 블록 접근 시간

제안하는 알고리즘의 블록 접근시간은 기존의 GMS 알고리즘에 비하여 약 10% 향상된 것을 볼 수 있다. 이것은 제안하는 알고리즘에서의 서버 메시지 부하가 GMS 알고리즘보다 적기 때문에 블록 접근시간이 약간 감소된 것이다. Hint-based 알고리즘의 경우 캐시 히트의 약 4.5% 정도가 잘못된 힌트정보로 인해 블록을 다시 검색하는 상황이 발생하였다. 따라서 재검색에 의한 시간지연의 결과로 블록접근시간이 더 길어지고 있다.

2.3 에이지 정보 관리비용 비교

캐시를 교체할 때, 비활동적인 노드를 선택하기 위해 협력캐시 알고리즘에서는 에이지 정보를 사용하는데 이때 사용되는 메시지 수를 에이지 정보 관리비용이라 정의한다. 캐시크기는 한정되어 있기 때문에 캐시의 블록 교체가 매우 빈번하게 발생하며 이 때 사용되는 메시지 수를 감소시키면 요청에 대한 응답 시간을 감소시켜 성능 향상의 중요한 역할을 한다. [그림 9]는 기존 알고리즘들과 제안하는 알고리즘의 에이지 정보 관리비용을 비교한 그림이다.

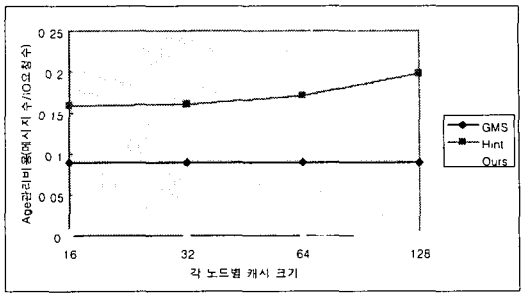


그림 9. 에이지 정보 관리 비용

제안하는 알고리즘의 경우 캐시 교체시 비활동적인 노드를 선택하기 위해 부가적으로 메시지를 보내어 에이지 정보를 가져오는 것이 아니라, 블록 검색이나 블록 전송으로 인하여 메시지를 교환하게 될 때, 노드의 에이지 정보를 첨부함으로써 메시지를 줄이는 방법을 사용하므로 기존의 알고리즘에 비해 약 20배~40배 정도 높은 성능을 보이고 있다.

2.4 일관성 유지를 위한 메시지 수 비교

기존 알고리즘은 캐시 내용이 변경될 때마다 일일이 다른 복사본들의 내용을 변경시켜 주고 있으나 제안하는 알고리즘은 LWI 방법을 사용하여 캐시의 일관성 유지를 위해 필요한 추가적인 메시지를 줄이므로 약 4배정도 높은 성능을 보이고 있다. [그림 10]은 기존 알고리즘과 제안하는 알고리즘의 일관성 유지를 위한 메시지 수를 비교한 그림이다.

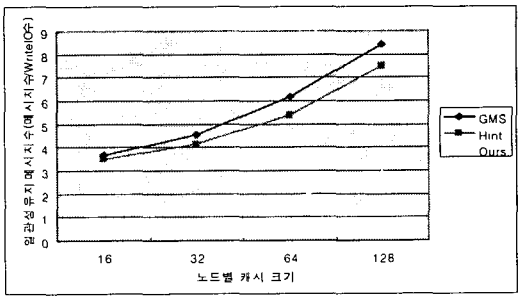


그림 10. 일관성 유지 메시지 수

V. 결 론

본 논문에서는 캐시를 중앙에서 관리하는 서버 없이 전체 시스템의 캐시 효율성을 향상시킬 수 있는 분산 파일시스템의 협력캐시 알고리즘을 설계하였다. 기존의 협력캐시 알고리즘 중에 모든 캐시정보를 중앙 서버가 제공해주고 있는 알고리즘의 경우 서버에 병목현상을 초래하고 있고, 캐시정보를 여러 노드로 분산시켜 관리하고 있는 알고리즘의 경우 캐시정보를 유지하는데 비용이 많이 드는 단점을 가지고 있다.

제안한 협력캐시 알고리즘에서는 중앙에서 관리하는 서버 없이 캐시정보를 관리하여 전체 시스템의 캐시 효율성을 향상시키고 있으며, 글로벌 캐시적중률과 로컬 캐시 적중률을 동시에 만족시킨다. 또한 비용은 감소시키면서 가치있는 데이터를 우선적으로 유지하기 위한 방법에 초점을 두어 캐시교체 수행을 효율적으로 처리하도록 설계하였고, 노드가 항상 최신의 캐시 데이터를 사용하도록 일관성을 유지하는 알고리즘으로 설계하였다.

제안한 협력캐시 알고리즘은 GMS 알고리즘과 Hint-based 알고리즘과 비교 평가하였다. 성능평가 결과에서 보듯이 캐시 적중률 면에서는 제안하는 알고리즘이 GMS 알고리즘과 Hint-based 알고리즘과 비슷한 성능을 보였으나 블록 접근시간 측면에서는 약 10% 성능 개선을 보이고 있으며 에이지 정보 관리비용은 20배~40배 정도의 상당히 많은 비용을 감소시키고 있다. 또한 일관성 유지를 위한 비용면에서도 기존 알고리즘에 비해 약 4배 정도 성능향상을 보이고 있다. 따라서 본 논문에서 제안하는 캐시알고리즘이 기존의 알고리즘에 비하여 효율적인 알고리즘임을 알 수 있다.

향후 연구 방향은 최근의 유사한 연구에 대한 추가적인 분석 및 본 논문에서 제안한 알고리즘과의 좀 더 다양한 측면에서의 성능 평가를 수행할 것이다. 또한 제안한 알고리즘의 블록 접근 시간을 좀 더 감소시킬 수 있도록 알고리즘을 수정 보완하는 것이다.

참 고 문 헌

[1] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. "Cooperative Caching : Using Remote Client Memory to improve File System Performance," In Proceedings of the 1st Symposium in Operating System Design and Implementation, pp.267-280, November 1994.

[2] W. H. Ahn, S. H. Park, and D. Park. Efficient Cooperative Caching for File Systemsin Cluster-Based Web Servers. In Proc. of IEEE International Conference on Cluster Computing, pp.326-334, Chemnitz, Germany, November 2000.

[3] Prasenjit Sarkar, John Hartman "Efficient Cooperative Caching using Hints" In Proceedings of the 2nd Symposium on Operating Systems Design and Implementation. pp.35-46, USENIX, October 1994.

[4] Michael J. Feeley, William E, Morgan, Frederic H.Pighin, Anna R. Karlin, and Henry M.Levy "Implementing Global Memory Management in a Workstation Cluster," In Proceedings of the 15th Symposium on Operating System Principles, pp. 201-212, December 1995.

[5] Menaud J, Issarny V, Banatre M. "A Scalable and Efficient Cooperative System for Web Caches," IEEE Concurrency, 8(3) pp.56-62 2000.

[6] Maru G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K.Ousterhout. "Measurements of a Distributed File Systems," In Proceedings of the 13th Symposium on Operating Systems Principles, pp.198-212, October

1991.

[7] T. Cortes and J. Labarta. Linear Aggressive Prefetching "A Way to Increase the Performance of Cooperative Caches," In Proc. of the 10th Symposium on Parallel and Distributed Processing, pp.46-54, San Juan, Puerto Rico, April 1999.

저 자 소 개

이 석 재(Seok-Jae Lee) 정회원

- 2000년 : 충북대학교 정보통신공학과(공학사)
- 2002년 : 충북대학교 정보통신공학과(공학석사)
- 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : DBMS, Real-time DBMS, XML, 자료저장시스템 등

박 새 미(Sae-Mi Park) 준회원

- 2002년 : 충북대학교 컴퓨터공학과(공학사)
- 2004년 : 충북대학교 정보통신공학과(공학석사)
- 현재 : LG전자

<관심분야> : DBMS, XML, Multi-DBMS 등

송 석 일(Seok-Il Song) 정회원

- 1998년 : 충북대학교 정보통신공학과(공학사)
- 2000년 : 충북대학교 정보통신공학과(공학석사)
- 2003년 : 충북대학교 정보통신공학과(공학박사)
- 현재 : 충주대학교 컴퓨터공학과 전임강사

<관심분야> : DBMS, 멀티미디어 데이터베이스, XML, 정보검색 등

유 재 수(Jae-Soo Yoo) 종신회원

- 1989년 : 전북대학교 컴퓨터공학과(공학사)
- 1991년 : 한국과학기술원 전산학과(공학석사)
- 1995년 : 한국과학기술원 전산학과(공학박사)
- 1995년~1996년 : 목포대학교 전산통계학과 전임강사
- 1996년~현재 : 충북대학교 전기전자컴퓨터공학부 부교수

<관심분야> : 데이터베이스시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등

이 장 선(Jang-Sun Lee) 정회원

- 1983년 : 경북대학교 전산학과(학사)
- 1985년 : KAIST 전산학과(석사)
- 1997년 : Syracuse Univ., Computer Science(박사)
- 1985 : 한국 전자 통신연구원(ETRI) 책임연구원
- 2000년~현재 : 매크로임팩트(주) 대표이사

<관심분야> : 병렬입출력, 클러스터파일시스템, SAN, 자료저장시스템, 운영체제 등