전역적 메모리에서의 캐시 일관성에 관한 연구

진연호[0] 김은경 정병수
경희대학교 전자계산공학과
{semipro, jlikek}@jupiter.kyunghee.ac.kr jeong@nms.kyunghee.ac.kr

A Study on the Cache Consistency in Global Memory

Yeon-Ho Jin[0] Eun-Kyung Kim Byeong-Soo Jeong
Dept. of Computer Engineering, Kyunghee University

요 약

최근의 네트웍 환경에서는 멀티미디어 서비스와 대용량의 파일을 이용하는 어플리케이션의 증가로 인해 이를 충족시킬 수 있는 저장 장치가 요구되고 있는 실정이며 이러한 저장 장치를 이용한 분산 환경의 네트웍 파일 시스템이 필수적이 되었다. 실제로 ATM, Fast switched LAN, Fibre channel 같은 고속의 네트웍 발달로 인해 분산 환경의 네트웍 파일 시스템에서 디스크를 액세스하는 속도보다 오히려 네트웍으로 연결된 원격지의 메모리를 액세스하는 것이 더 빨라졌다. 따라서 지역 디스크 캐싱 기법이 분산 환경의 네트웍 저장 시스템으로 적용되면서 전역적 메모리를 관리하는 것과 원격지간의 캐시 일관성 문제(cache consistency)를 고려하지 않을 수 없게 되었다. 본 논문에서는 분산 환경의 캐싱 기법을 살펴보고 전역적 메모리의 캐시 일관성 문제를 다루면서 이에 대한 설계방안 및 앞으로의 연구 방향을 제시한다.

1. 서론

최근 들어 멀티미디어 데이터 서비스 및 대용량의 파일을 이용하는 애플리케이션의 증가로 여러 가지 기법을 적용한 분산 환경의 네트웍 저장 시스템이 연구되어 왔다. 이는 지난 20년 간의 전형적인 컴퓨팅 시스템의 틀을 벗어난 것이라 할 수 있다. 즉 [표1]처럼 메인 프레임급의 저가의 고성능 프로세서와 값싼 저장장치, 그리고 고속 광역의 네트웍이 컴퓨팅 패러다임을 새롭게 바꾸게 된 것이다. 이러한 패러다임의 변화가 분산 네트웍 파일 시스템의 연구로 확장되었으며 네트웍을 통하여 많은 자원을 공유하게 되었다.

H/W Parameter	1995 Baseline	증가율
Disk latency	12 ms	10%/year
Disk bandwidth	6-9 MB/s	20%/year
Disk cost/capacity	$0.24/MB	100%/year
Network latency	1 ms	20%/year
Network bandwidth	20 MB/s	45%/year
Processor performance	100SPECInt92	55%/year
Memory bandwidth	30-70 MB/s	40%/year
Memory cost/capacity	$31/MB	45%/year

[표1] 하드웨어의 성능 경향

이와 같은 사실은 지금까지 연구되어온 NFS나 AFS, 그리고 Sprite 시스템[1]에서 네트웍을 통한 캐싱이 상당히 중요한 역할을 하고 있음을 시사하고 있다. 특히 AFS는 파일의 전체 캐싱을 이용하여 워크스테이션을 5000 여개 이상 확장할 수 있도록 확장성(Scalability)에 목표를 두고 있다[2].

본 논문은 최근에 연구되고 있는 고속의 네트웍을 이용한 대용량 저장 시스템을 대표하는 SAN(Storage Area Network)이나 NAS(Network Attache Storage) 와 같은 네트웍 저장 시스템을 기반으로 연구하였으며 다음과 같이 구성된다. 2장에서는 기존에 연구된 캐싱과 캐싱된 공유객체의 일관성 유지 기법에 대해 설명하며 3장에서는 본 논문에서 제안하는 서버 부담을 줄이고 일관성을 유지할 수 있는 정책과 설계방안을 기술하며 4장에서는 결론을 맺는다.

2. 관련연구

2.1 협력 캐싱(Cooperative Caching)

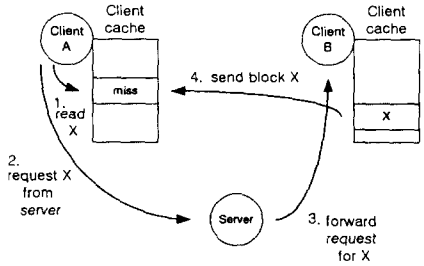
단일 머신(machine)에서 디스크 액세스를 줄이기 위해 메모리 캐시를 이용하는 것처럼 최근의 분산 환경의 네트웍 저장 시스템은 노드들이 고속의 네트웍에 연결되어 있어서 캐싱 실패(caching miss)시 로컬 디스크를 액세스하지 않고 인접한 노드의 메모리 캐싱을 이용하도록 한다. [표2]는 분산 환경의 네트웍을 통한 캐싱이 로컬 디스크의 캐싱보다 훨씬 더 빠름을 보여주고 있다.

	Local Memory	Local Disk	Ethernet		77Mbit/s ATM	
			Remote Memory	Remote Disk	Remote Memory	Remote Disk
Mem. copy	250	250	250	250	250	250
Net. Overhead & Data			400 6,250	400 6,250	400 800	400 800
Disk		14,800		14,800		14,800
Total	250	15,050	6,900	21,700	1,450	16,250

[표2] 캐싱 실패 시 걸리는 시간 (단위 *us*)

협력 캐싱 기법은 한 노드의 캐시 교체(replacement)가 일어날 때 유휴 클라이언트의 캐시 메모리를 사용하는 Direct Client cooperation 과 서버를 통해 각 노드의 캐시를 전역적으로 관리하는 Greedy Forwarding, 그리고 캐시를 local 영역과 global 영역으로 나누어 각 노드에 있는 캐시 내의 객체들을 전역적으로 조정(coordination)하는 Centrally Coordination Caching 등으로 크게 분류되어 연구되어 왔다[3].

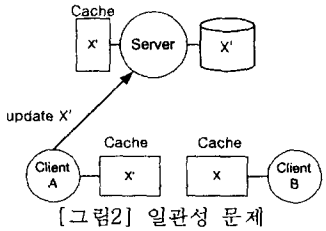
협력 캐싱은 시스템의 성능과 매우 밀접한 관계가 있어서 네트웍으로 연결된 각 노드의 캐시가 어떤 방법으로 조사(lookup), 교체(replacement)되는 지가 그 시스템의 성능과 확장성에 관건이 된다. [그림1]은 협력 캐싱 기법중의 하나인 Greedy Forwarding 기법을 설명하고 있다[3,4].



[그림1] Greedy Forwarding

2.2 캐싱 일관성(Caching Consistency)

분산 환경을 통해 각 노드들이 캐싱을 하다보면 공유 객체를 가지게 되며, 공유 객체를 가진 노드가 그 객체에 대해 갱신을 하는 경우가 생기게 된다. 따라서 이렇게 갱신하는 노드 이외의 예전 공유 객체를 가진 각 노드들은 일관성 문제를 가지게 된다. 이러한 일관성 문제는 분산 네트웍 파일 시스템뿐만 아니라 분산 DBMS 에서도 일어나는 문제이며 특히, 공유 메모리 멀티 프로세서 구조에서는 프로세서의 성능과 신뢰도에 중요한 요소가 된다. [그림2]는 이러한 일관성 문제를 보여주고 있다.



[그림2] 일관성 문제

캐싱의 일관성 문제는 노드간의 공유 객체의 갱신에서 발생되며 이 문제를 가장 쉽게 해결하는 방법은 각 노드에서 변경된 객체(dirty object)를 참조하는 복사본을 없애고 디스크로부터 갱신된 객체를 가져오는 것이다. 대부분의 파일 시스템에서는 파일 연산 중 쓰기 연산(write operation) 이 상당 부분 차지하기 때문에 이것은 네트웍 저장 시스템에서 치명적인 성능 저하를 가져오는 요인이 된다. 따라서 기존에 연구된 분산 파일 시스템의 쓰기 정책(write policy)과 일관성 유지 방식을 살펴보도록 한다.

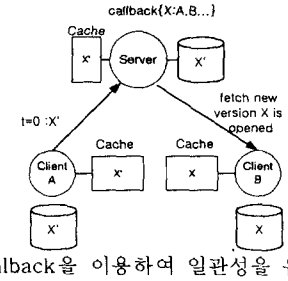
2.2.1 쓰기 정책(Write Policy)

앞 절에서 기술한 변경된 객체(dirty object)를 서버 내의 캐시 또는 디스크로 쓰기 연산을 수행하는 것은 시스템 내에 지연(latency) 또는 병목현상(bottleneck)을 주므로 분산 네트웍 파일 시스템 설계 시 쓰기 정책은 시스템 성능에 있어서 중요한 요소가 된다. 가장 간단한 전략은 캐시 내의 객체를 갱신 했을 때 바로 디스크로 기록하는 것이다. 이 바로 쓰기(write-through) 정책은 시스템이 고장나 멈춘다 해도 캐시 내의 객체 소실량이 적으므로 신뢰도가 매우 높다. 그러나 잦은 디스크의 액세스로 인해 쓰기 성능이 떨어지는 단점이 있다. NFS가 이러한 바로 쓰기 정책을 사용한다.

지연 쓰기(write-delay) 전략은 위와 반대로 캐시 내의 객체를 변경한 후 일정 시점에서 변경된 객체(dirty object)를 디스크로 쓴다. 이 정책은 쓰기 연산이 어느 시점까지는 메모리 내에서만 일어나므로 매우 빠른 쓰기 연산을 가진다. 또한 캐시 내에서 빈번한 쓰기 작업이 일어나거나 좀 전에 썼던 것이 사용자 임의로 지워지는 경우 디스크로의 불필요한 쓰기 횟수가 줄어드는 장점이 있다. Sprite 시스템에서는 30초를 지연하여 쓰고 있으며, 변형된 지연 쓰기로는 AFS 시스템의 폐쇄시 쓰기(write on close) 정책이 있다[1,2].

2.2.2. 일관성 유지(Consistency maintenance)

분산 네트웍 파일 시스템에서 캐시의 일관성을 어떻게 유지하느냐에 따라 네트웍을 통한 서버 또는 캐싱 매니저의 접근 횟수를 줄여 확장성(scalability)을 향상시킬 수 있다. 공유 객체에 대한 각 노드들의 일관성 유지 방법은 크게 클라이언트 주도의 접근 방법(Client-initiated approach)과 서버 주도의 접근 방법(Server-initiated approach)의 두 가지로 나뉜다[5]. 클라이언트 주도의 접근 방법은 클라이언트가 항상 서버와의 접촉을 통해 일관성 검사를 하여 지역 캐시 내의 객체가 서버 내의 마스터 복사본과 일치하는지 검사한다. 한편 서버는 임의의 객체를 어느 클라이언트가 사용하고 있는지에 관한 정보를 저장하지 않으므로 공유 객체가 갱신되었을 때 이를 참조하는 다른 노드에게 변경된 사실을 알릴 방법이 없다. 서버 주도의 접근 방법은 서버가 현재 사용 중인 객체들의 정보를 유지함으로써 각 노드들이 캐싱 하고 있는 공유 객체의 일관성을 유지하도록 한다. Sprite 시스템의 경우는 객체를 쓰기 방식으로 캐싱할 때마다 버전 번호(version Number)를 증가시켜 일관성을 유지하는 반면[1] AFS 시스템은 콜백(Callback)을 통해 일관성을 유지한다. [그림3]은 서버 주도의 접근 방법을 사용하는 AFS시스템의 일관성 유지를 설명하고 있다.

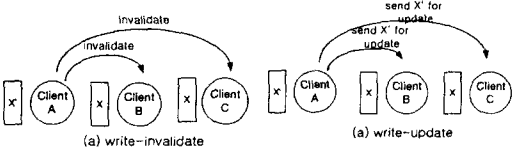


[그림3] Callback을 이용하여 일관성을 유지하는 AFS

3. 일관성(Consistency) 설계 방안

본 논문의 고속 네트웍 저장 시스템의 일관성 설계를 10위하여 몇 가지 사항으로 나누어 살펴본다. 먼저 분산 네트웍 환경의 특성인 확장성을 고려하여 네트웍의 데이터나

메시지 전송 횟수를 최소화 하고 서버 부하를 분산시켜야 한다. 이를 위하여 모든 노드들의 전역적 메모리를 관리할 수 있는 캐시 매니저를 두고 디스크에 저장된 객체를 분할하여 담당하게 함으로써 캐시 일관성 유지를 할 수 있다. 본 논문은 이러한 서버 주도의 접근 방법을 채택하는데 그 이유는 클라이언트 주도형에서 발생하는 서버와의 불필요한 교신 메시지로 인한 시스템의 확장성 문제를 피할 수 있기 때문이다. 서버 주도형은 서버가 관리해야 할 캐시 내의 객체 정보가 많다는 단점이 있지만 여기서는 디스크 내의 모든 객체들을 분담하여 관리하는 캐시 매니저를 두기 때문에 서버의 부담이 줄어든다. 다음은 캐시 내의 공유 객체가 변경되었을 경우 그 공유 객체를 참조하는 다른 노드들에게 어떻게 할 것인가의 프로세싱 결정이다. 본 논문에서는 기본적으로 쓰기-무효화(write-invalidate)와 쓰기-갱신(write-update)방식 중에 쓰기-무효화 방식을 채택하고 무효화 처리를 서버(매니저)가 아닌 각 노드들이 자발적으로 하는 기법[6]을 사용하면서 Sprite 방법과 유사한 지연 쓰기 정책을 도입한다.



[그림4] 공유 객체 변경 시 처리 방법

따라서 매니저는 객체에 대한 쓰기 연산 시 필요한 정보를 저장 및 처리하여 이를 요청한 노드로 정보를 보내어 일관성을 유지하도록 한다. 본 논문에서는 다음과 같은 방법으로 매니저와 노드간에 메시지 전송을 통해 일관성을 유지하고자 한다. 기본적으로 매니저에 접근하여 응답 메시지를 받을 때는 매니저가 유지하고 있던 무효화 집합을 편승(piggy-backing)하여 받는다. 그리고 한 노드 A 가 객체를 쓰기 또는 변경하고자 할 때는 그 객체에 대한 쓰기 권한의 토큰을 매니저로부터 얻는다. 만일 그 객체를 다른 노드 B 가 읽고자 한다면 토큰을 가지고 있던 노드는 매니저로부터 토큰을 박탈당하고 그 객체의 복사본을 매니저를 통해 요청된 노드 B 로 전해주며 노드 A 의 공유 객체는 읽기 모드로 변경된다.

Client
읽기 실패시
요구하는 객체와 자신의 무효화 집합을 획득.
요구한 객체는 읽기 모드로 세트.
쓰기 실패시
요구하는 객체와 자신의 무효화 집합을 얻고 토큰을 요구.
요구한 객체는 쓰기 모드로 세트.

Server
읽기 실패로 접근해온 노드를 위해
요구된 객체에 token이 부여된 다른 노드가 있으면 그 노드의 토큰을 박탈.
노드 이름이 붙은 무효화 집합과 객체를 전송.
쓰기 실패하여 접근해온 노드를 위해
요구된 객체에 token이 부여된 다른 노드가 있으면 그 노드의 토큰을 박탈.
요구된 객체를 캐싱하고 있는 노드들의 이름이 붙은 각각의 무효화 집합에 요구된 객체 이름을 추가.
요구된 객체와 요구한 노드의 이름이 붙은 무효화 집합을 전송.

[그림4] 노드와 서버간의 처리 방법

다시 노드 A 가 그 공유 객체를 변경하고자 한다면 토큰이 없으므로 다시 매니저로 접근하여 토큰을 얻는다. 이 때 서버는 캐싱 된 객체들의 정보를 살펴보고 서버 내에 노드 B 의 이름이 붙은 무효화 집합을 갱신한다. 이 무효화 집합은 나중에 노드 B 가 매니저와 통신할 경우에 얻게 되므로 자발적인 무효화를 실행하게 된다. 이렇게 함으로써 항상 서버가 공유된 객체를 가진 모든 노드들에게 일일이 무효화 처리를 하는 작업을 하지 않더라도 그 객체를 가진 노드들이 자발적인 무효화 처리를 하여 자신의 캐시에서 객체를 없애버린 후 디스크나 서버로부터 새로운 갱신된 객체를 얻어 일관성을 유지하도록 한다. 더욱더 강한 일관성을 유지하기 위해서 캐시 내의 공유 객체가 토큰을 박탈 당할 때에 디스크로 쓰기 작업을 바로 수행하거나 또는 매니저가 일정 시간에 무효화 집합을 조사하여 해당되는 노드에게 그 집합을 전해주도록 설계한다. [그림4]는 각 노드와 매니저가 해야 할 역할을 설명하였고 [표3]은 그 역할을 기반으로 간단한 과정을 도시한 것이다.

step	Node A	Node B	Invalset[A]	Invalset[B]	Token & Disk
1	W(x) 0		Empty	empty	Grant x
2		W(y) 0			Grant y
3	R(y) 0				Revoke & sink y
4		R(x) 0			Revoke & sink x
5	W(x) 1			Add x	Grant x
6		W(y) 1	Add y	Remove x	Grant y
7	R(y) 0				Revoke & sink y
8		R(x) 1			Revoke & sink x

[표3] 일관성 유지 과정 (W/R: 쓰기/읽기, x/y: 객체)

4. 결론 및 향후 과제

본 논문에서는 분산 환경의 고속 네트웍 저장 시스템에서 캐시 일관성을 유지하는 데에 있어서 주요 문제가 되는 시스템 형태나 캐싱 기법, 쓰기 정책, 일관성 유지 기법을 살펴보았다. 또한 제안된 시스템의 확장성을 고려하여 네트웍상의 메시지를 최소화하고 서버의 부담을 덜어주기 위해 디스크에 저장되어 있는 객체들을 나눠 전담하는 캐시 매니저를 설정하였다. 그리고 매니저를 통해 변경된 공유 객체의 무효화 처리를 클라이언트에서 자발적으로 하는 설계 방안을 제안하였다. 향후 연구로는 이를 바탕으로 다양한 환경의 시뮬레이션 결과를 산출 및 비교하여 SAN(Storage Area network) 이나 NAS(Network Attached Storage) 환경에서 커널 레벨의 버퍼 캐시 변경이 가능한 리눅스 기반의 캐시 매니저 설계를 할 것이다.

5. 참고 문헌

[1] M. Nelson, B. Welch, and J. Ousterhout. "Caching int the Sprite Network File System." ACM Trans. on Computer Systems, 6(1), February 1988.

[2] R.Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. "Design and Implementation of the Sun Network File System." In Proc. of the Summer 1985 USENIX, pages 119-130, June 1985.

[3] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, Randolph Y. Wang. "Severless Network File Systems." 15th ACM Symposium on Operating Systems Principles, December 1995 and ACM Transaction on Computer Systems, 1996.

[4] Prasenjit Sarkar and John Hartman. "Hint-based Cooperative Caching." University of Arizona, 1998. 2.

[5] A. Silberschatz. "Operating System Concepts 3rd edition.", Addison-Wesley publishing company, 1993.

[6] M. Ahamad, R. kordate. "Scalable Consistency Protocols for Distributed Services." IEEE Transaction on parallel and Distributed System, VOL.10, NO.9, September 1999.