이동 컴퓨팅 환경에서 캐쉬 관리를 위한 TWB 기법

(Trickle Write-Back Scheme for Cache Management in Mobile Computing Environments)

김 문 정 † 엄 영 익 ††
(Moon Jeong Kim) (Young Ik Eom)

요 약 최근 사용자가 위치를 이동하는 도중에도 네트워크에의 연결을 계속해서 유지할 수 있도록 하는 이동 컴퓨팅 환경에 대한 연구가 이루어지고 있다. 이러한 이동 컴퓨팅 환경을 위해 설계되고 구현되어야 할 요소 중의 하나가 이동 호스트를 고려한 분산 화일 시스템이며 이에는 여러 가지 설계 사항들이 존재한다. 이러한 설계 사항들 중에는 제한된 대역폭 상에서의 통신량 문제와 공유 화일에 대한 여러 사용자들의 수정 충돌(update conflict)로 인하여 발생되는 데이타 일치성 문제가 있다. 본 논문에서는 이동 컴퓨팅 환경에서 이동 클라이언트의 캐쉬 관리를 위하여 약한 연결상태를 이용하는 TWB(Trickle Write-Back) 기법을 제안한다. 본 제안 기법은 불필요한 write-back을 지연시키되 일정시간마다 중간 상태를 write-back 함으로써, 이동 컴퓨팅 환경에서 중요한 자원인 대역폭을 절약함과 동시에 그로 인한 디스크 낭비와 단절 상황 시에 발생될 위험 부분을 축소하도록 한다. 또한 기존의 분산 화일 시스템 구조를 충분히 이용하면서 사용자에게는 가능한 한 투명성을 제공한다.

Abstract Recently, studies on the mobile computing environments that enable mobile hosts to move while retaining its network connection are in progress. In these mobile computing environments, one of the necessary components is the distributed file system supporting mobile hosts, and there are several issues for the design and implementation of the shared file system. Among these issues, there are problems caused by network traffic on limited bandwidth of wireless media. Also, there are consistency maintenance issues that are caused by update-conflicts on the shared files in the distributed file system. In this paper, we propose TWB(Trickle Write-Back) scheme that utilizes weak connectivity for cache management of mobile clients. This scheme focuses on saving bandwidth, reducing waste of disk space, and reducing risks caused by disconnection. For such goals, this scheme lets mobile clients write back intermediate states periodically or on demand while delaying unnecessary write-backs. Meanwhile, this scheme is based on the existing distributed file system architecture and provides transparency.

1. 서 론

현재 컴퓨터 기술은 이동중인 사용자를 지원하는 방향으로 발전하고 있다. 이에 따라 최근 사용자의 이동성을 지원하는 여러 기술들에 대한 연구가 활발히 진행중이다. 이는 무선망을 이용하여 노트북, 팜탑과 같은 휴대용 컴퓨터를 통해 사용자의 이동 중에도 네트워크 서비스를 제공받도록 지원하는 기술이며, 이러한 기술을 기반으로 한 컴퓨팅 환경을 이동 컴퓨팅 환경(mobile computing environment)이라 한다.

이러한 이동 컴퓨팅 기술이 기존의 고정 네트워크를 통해 지원하던 모든 서비스를 거의 흡사한 질로 사용자에게 제공하기에는 많은 제약점들이 존재한다. 우선 무

· 이 논문은 성균관대학교의 1997년도 성균학술연구비에 의하여 연구되었음
† 비 회 원 : 성균관대학교 전기전자및컴퓨터공학부
tops@ece.skku.ac.kr
†† 종신회원 : 성균관대학교 전기전자및컴퓨터공학부 교수
yieom@ece.skku.ac.kr
논문접수 : 1999년 1월 4일
심사완료 : 1999년 11월 1일

선통신 환경은 기존의 유선통신 환경에 비해 데이타의 손실, 왜곡, 도난 등의 위험성이 크다. 또한 이동 컴퓨팅 환경에서 디지털 데이타 서비스를 제공하는 경우 전송 중의 작은 양의 손실이 전송된 모든 자료를 불필요한 자료로 만들 위험도 크다. 특히 무엇보다도 이동 컴퓨팅 환경에서는 이동성(mobility), 휴대성(portability), 그리고 저 대역폭(low bandwidth)의 문제가 두드러진다.

이와 같은 이동 컴퓨팅 환경과 관련한 여러 가지 제약에도 불구하고 최근에는 이동 컴퓨팅 환경이 단순한 통신 서비스뿐만 아니라 데이타 서비스 및 분산 화일 시스템을 확장하는 형태의 화일 서비스를 제공하는 형태로까지 발전하고 있다[1]. 무선 통신 하부구조 상에서 이러한 데이타 서비스와 화일 서비스를 제공하기 위해서는 무선 대역폭을 효율적으로 사용하기 위한 기법이 요구되며, 이러한 서비스를 제공하는 과정에서 발생하는 통신 트래픽을 줄이거나 가용한 통신 대역폭을 효과적으로 이용하는 기법의 연구가 더욱 중요해 지고 있다.

이동 호스트를 지원하는 분산 화일 시스템 환경에서는 클라이언트에서 수정된 데이타의 write-back을 지연시킴으로써 제한된 대역폭 상에서의 통신량을 줄이는 것이 시스템 성능에 많은 잇점을 줄 수 있다는 사실이 여러 연구를 통해 밝혀져 있다[2, 3, 4, 5]. 이러한 지연 write-back 기법들의 사용은 이미 많은 화일 시스템들이 긍정적인 복제(optimistic replication) 기법을 사용함으로써 어느 정도 일치성(consistency)과 가용성(availability)간의 균형을 염두에 두고 있기 때문에 가능하다.

앞으로 이동 컴퓨팅 환경이 일반화되고 이에 따라 컴퓨터 시스템들이 보다 소형화될수록 더욱 많은 화일들이 공유의 대상이 될 것으로 기대된다. 특히, 이러한 시스템들의 응용 환경에서는 쓰기 공유가 비교적 드물다는 사실이 알려져 있으며[6, 7, 8, 9], 따라서, 공유 화일 시스템을 사용하는 경우 공유된 화일들 중 상당수가 동시에 여러 사용자들에 의해 접근되기보다는 일정 기간 동안 한 개인에 의해 접근될 가능성이 크게 된다. 이러한 점으로 인해 지연 write-back 기법이 시스템 성능을 높이는 효과를 보이게 되는 것이다.

본 논문에서 제안하는 기법은 한 사용자에 의해 수정된 화일이 가까운 시간 내에 반복적으로 재수정되거나 또는 삭제되는 경우에 보다 유용한 기법이다. 이 기법은 화일의 수정된 내용에 대해 화일 닫기 명령이 수행될 때마다 write-back 하지 않고 여러 번의 닫기 명령 후 write-back 하도록 함으로써 클라이언트의 부담과 협소한 대역폭 상에서의 통신 트래픽을 감소시키는 기법이다. 사용자의 개입 없이 write-back을 지연시키며, 화일 닫기와 관계없이 필요한 시간에 write-back을 수행함으로써 단절 상태로 인해 발생할 수 있는 위험을 최소화하고, 서버로부터도 write-back을 명령받을 수 있도록 하여 write-back 지연으로 인하여 추가로 발생할 수 있는 쓰기 충돌의 기회 증가를 방지하는 기법으로, TWB(Trickle Write-Back) 기법을 제안한다.

또한, 본 제안 기법은 항상 write-back을 일정 시간 동안 지연시키는 기존 기법과 달리 수정된 화일에 대해 write-back을 지연시키는 동안 이동 컴퓨팅 환경의 특징인 약한 연결 상태를 이용하여 주기적으로 조금씩 write-back을 시도함으로써 기록 화일로 인한 지역 디스크 낭비와 단절시의 위험을 최소화한다. 따라서, write-back의 주기를 네트워크의 상태에 따라 동적으로 설정할 수 있는 경우에 보다 유용한 기법으로 사용될 수 있을 것이다.

2. 관련 연구

2.1 Coda 화일 시스템

CFS(Coda File System)는 미국 CMU(Carnegie-Mellon University)에서 개발한 분산 화일 시스템으로 기존의 분산 화일 시스템들과 달리 어느 정도 호스트의 이동성을 지원한다[10, 11]. CFS에서 클라이언트 측의 지역 디스크를 관리하고 서버와의 통신을 담당하는 부분을 Venus라 하며, 서버들간의 데이타 일치성을 유지시켜 주며 클라이언트와의 통신을 담당하는 부분을 Vice라 한다.

CFS의 궁극적인 목표는 데이타의 가용성(availability)을 높이는데 있다. 이를 위해 서버 복제(server replication) 기법과 비접속 연산(disconnected operation) 기법을 제공한다. 우선 순위 알고리즘을 이용한 캐슁 기법을 사용하고, 캐쉬 관리를 위하여 LRU 기법과 사용자의 개입을 요구하는 HDB(Hoard DataBase)를 함께 사용하며, 주기적으로 캐슁된 자료들이 우선 순위에 일치하는가를 확인한다. 또 콜백-기반 캐쉬 일치성(callback-based cache coherence) 기법을 사용하여 캐슁되어 있는 자료가 다른 사용자에 의해 수정되는 경우 서버로부터 클라이언트로 화일의 수정 여부를 통보하도록 한다. 또한 서버간 데이타 일치성을 보장하기 위해 CVV(Coda Version Vector) 개념을 사용한다. CFS에서 write-back을 위해 write-on-close 기법을 사용하며, 이는 클라이언트가 캐쉬한 화일의 수정한 내용을 화일 닫기 후에 write-back 하는 기법이다.

기존의 CFS에 추가로 제안되는 기법들 중의 하나인

Trickle reintegration이란 비접속 연산의 수행 결과를 재접속 상태에서 비동기적으로 서버에게 전송하는 기법이다[12]. Venus의 상태 중에 쓰기 비접속 상태(write disconnected state)를 추가하였으며, 이 상태에서는 물리적으로 재연결이 이루어진 후 바로 캐쉬 내의 모든 자료에 대해 일치성을 맞추지 않고 자동적으로 조금씩 맞추어 나가도록 하는데 이때 사용되는 기법이 Trickle reintegration이다. 이 상태에서 특정 응용에 의해 캐쉬 오류(cache miss)가 발생하는 경우에는 바로 서버와 접속하여 정상적인 서비스를 제공하고, 캐싱되어 있는 자료에 대해서는 기존의 에뮬레이션 상태(emulation state)에서와 같이 서비스하게 된다. 이 기법은 쓰기 비접속 상태 동안의 작업 내용을 모두 기록 화일(log file)에 기록하므로 비교적 장시간 지연이 가능하며 단절되기 전에 기록된 내용을 조금씩 전달하므로 기록 화일 최적화 과정의 오버헤드를 줄여준다. 이 기법 역시 네트워크상의 통신 트래픽을 줄이는데 주요 목표를 둔 기법이다.

2.2 Echo 화일 시스템

Echo 화일 시스템에서는 write-back 기법으로 write-behind 기법을 사용하며, 이 기법은 토큰(token)과 시간제한(time-out) 개념을 함께 사용한다[9]. 이 시스템에서 Echo 화일스토어(filestore) 서버는 토큰을 관리하며, 클라이언트에게 토큰을 지급한다. 클라이언트가 화일을 읽기 위해서는 읽기 토큰(read token)을 지급받아 소유하고 있어야 하고, 동일 화일을 수정(update)하기 위해서는 쓰기 토큰(write token)을 지급 받아 소유하고 있어야 하며, 일정 시간 후에 write-back을 수행한다. 만약 임의의 클라이언트가 어떤 화일의 쓰기 토큰을 가진다면, 다른 모든 클라이언트들은 동일 화일에 대한 읽기 또는 쓰기 토큰을 가질 수 없다. Echo 화일 시스템의 write-behind 기법에서는 클라이언트 응용이 화일 생성, 쓰기, 다시 읽기, 지우기 등의 연산을 완전히 클라이언트 내에서 수행하며, 서버의 개입이 전혀 없도록 하고 있다.

이동 컴퓨팅 환경에서는 단절 상태보다는 약한 연결 상태가 많이 나타나므로 CFS 등의 기존 분산 화일 시스템에서 염려했던 단절 시에 지역 디스크에 없는 화일들을 이용하는 문제는 크게 심각하지 않다. 지역 디스크에 없는 화일이 사용자에 의해 요구될 경우에 약한 연결 상태에서 클라이언트가 서버에게 해당 화일을 요청하고 서버는 비록 늦은 전송이겠지만 클라이언트에게 필요한 데이타를 전송해 줄 수 있는 것이다[13].

물론 이동 컴퓨팅 환경에서는 고정 네트워크 상에서보다 통신 지연과 짧은 끊김이 훨씬 자주 발생한다. 따라서 기존의 콜백 기법 외에도 버전 스탬프(version stamp)와 전체 화일 시스템상의 콜백을 유지하는 기법 등이 추가로 제안되고 있다[14].

3. Trickle Write-Back 기법

3.1 개요

본 논문의 제안 기법을 위해 두 가지 가정을 둔다. 우선 가까운 미래에 사용자가 접근할 가능성이 있는 화일들을 비접속 상태가 되기 전에 미리 캐쉬 내에 적재시켜 놓는 hoarding을 위해 사용자가 작성한 프로파일(profile)을 사용하며, 또한, 캐싱 정책으로는 우선 순위 기반의 기법을 사용한다고 가정한다.

서버에 공유된 화일을 클라이언트가 캐쉬하여 수정한 후 최신 화일을 다시 서버로 전송하는 것을 write-back이라 한다. 이러한 write-back을 지연시키는 경우의 잇점에 대해 기술한 연구 결과들이 다수 발표되어 있으며, 또한 이를 위한 구체적인 기법들이 제안되어 있다[2,3,4]. 기존에 제안된 write-back 지연 기법으로는 CFS에서와 같이 수정된 화일을 닫을(close) 때마다 write-back하거나 비접속 상태에서도 화일의 수정이 가능하게 하고 이렇게 수정된 화일을 재접속 시에 write-back 하는 정책이 있다[15]. 그러나 수정된 화일을 닫을 때마다 write-back 하는 정책은 가까운 시간내에 같은 사용자로부터 재수정되거나 삭제되는 경우 앞에 전송된 화일이 불필요해지게 되는 경우가 발생한다. 이러한 상황은 이동 컴퓨팅 환경이 더욱 확산되어 대부분의 개인적인 화일을 공유하게 되는 환경에서는 빈번히 발생되는 상황이 될 것이다. 또다른 지연 기법으로는 Echo 화일 시스템에서와 같이 수정된 화일에 대해 일정 시간동안 write-back을 지연시키는 기법이 있다[4]. 그러나 write-back을 일정시간 지연시키는 기법은 지연시키는 동안의 작업 내용을 기록하기 위해 지역 디스크를 사용해야 하므로 캐쉬를 위해 지역 디스크의 대부분이 사용될 수는 없도록 한다. 또한, 중에 다른 클라이언트에 의해 write-back이 수행되는 경우인 쓰기-쓰기(write-write) 충돌이 발생할 기회의 증가를 초래하게 되며, 이러한 충돌 현상은 지연 기법들이 가져올 잇점을 크게 감소시키게 된다.

본 논문의 제안 기법은 기존 기법중 CFS와 같이 수정된 화일의 닫기 명령 후 바로 write-back 하는 기법과는 달리 한 사용자에 의해 수정된 화일이 가까운 시간 내에 반복적으로 재수정되거나 또는 삭제되는 경우에 여러 번의 닫기 명령을 시행한 후 한 번만 서버로

write-back 함으로써 클라이언트의 부담과 협소한 대역폭 상에서의 통신량을 줄이도록 하는 기법이다. 또 Echo 화일 시스템과 같이 일정기간 write-back을 지연하는 기존 기법과는 달리 수정된 화일에 대해 write-back을 지연시키는 동안 이동 컴퓨팅 환경의 특징인 약한 연결 상태를 이용하여 주기적으로 조금씩 write-back 함으로써 기록 화일로 인한 지역 디스크 낭비와 비접속시의 위험을 최소화한다. 또한 서버로부터 write-back 명령을 받을 수 있게 함으로써 write-back을 지연시키므로써 추가로 발생 가능한 쓰기 충돌 현상을 방지하는 기법이다.

본 논문의 제안 기법은 한 사용자에 의해 수정된 화일이 가까운 시간 내에 반복적으로 재수정 되거나 또는 삭제되는 경우가 빈번한 경우에 유용한 기법이며, write-back의 주기를 네트워크의 상태에 따라 동적으로 설정할 수 있는 경우에 보다 유용한 기법으로 사용될 수 있을 것이다.

3.2 요구 사항

3.2.1 제안 시스템 모델

본 논문에서 제안하는 기법의 시스템 모델은 그림 1에서 보인다.

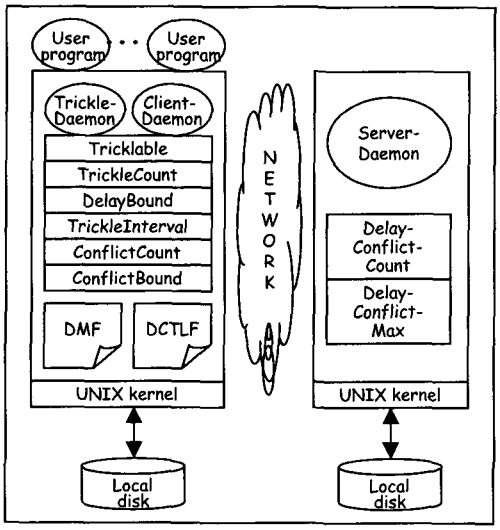


그림 1 본 제안 기법의 시스템 모델

본 제안 기법에서는 CFS에서와 같이 hoarding을 위해 사용자가 정의한 프로파일을 유지시키는 방법을 사용하며, 이를 위해 HoardDB라는 사용자 프로파일을 사용한다. 또한 캐슁과 관련하여 LRU 및 우선 순위 기반의 기법을 사용하며, 우선 순위가 높은 화일은 사용자에 의해 작업이 빈번한 화일일 확률이 높다는 사실을 이용한다. 즉, 우선 순위가 높은 화일이면서 사용자가 수정한 경우라면, 가까운 시간 내에 동일 사용자에 의해 재수정이 발생할 확률이 높다는 사실을 이용한다.

사용자의 개입 없이 서버의 허가를 얻어 클라이언트의 자발적인 write-back 지연이 이루어지며, 클라이언트마다 적정한 시간 간격을 두고 주기적으로 또는 필요한 때에 write-back을 실행함으로써 약한 연결성(weak connectivity)을 이용하고 단절시의 손실을 최소화하도록 한다. 둘 이상의 클라이언트에 의해 동시에 자주 접근되는 화일이나 네트워크의 단절이 잦은 클라이언트의 경우에는 본 논문에서 제안하는 기법이 오히려 큰 오버헤드를 발생시킬 수 있으므로, 전자의 경우는 서버 측에서, 후자의 경우는 클라이언트 자체 내에서 본 논문의 제안 기법을 적용하지 않도록 할 수 있게 한다. 3.2.2 절과 3.2.3 절에서는 클라이언트와 서버 내에 요구되는 데이타 구조에 대해 설명한다.

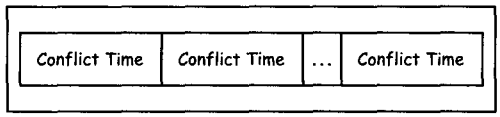
3.2.2 클라이언트 측 데이타 구조

본 논문에서 제안하는 TWB 기법을 위해서 클라이언트 측에 필요한 데이타 구조를 표 1과 그림 2에서 보인다. 표 1에서는 클라이언트 측에 필요한 각종 변수들을 보여주고 있다. 그림 2-(a)에서는 현재 지연중인 화일을 관리하는 DMF(Delay Manage File)의 각 엔트리 형식을 나타낸다. 또한, DCTLF(Disconnection Conflict Time Log File)를 두고 단절 후 재복구 시에 그 동안 서버와 클라이언트간에 충돌이 발생하였음이 밝혀질 경우 당시의 시간을 기록하도록 한다(그림 2-(b)). 그 외에도 hoarding을 위해 사용자에 의해 작성되는 프로파일인 HoardDB를 사용하여 사용자가 자주 사용할 화일들에 대해 그 우선 순위를 사용자 임의로 기록, 유지시킬 수 있도록 한다.

표 1 클라이언트 측 데이타 구조

DelayBound	Trickle write-back을 요구하게 될 우선 순위의 경계치
TrickleInterval	서버의 요구 없이 자발적으로 write-back 하게 될 시간 간격
ConflictCount	정해진 기간 동안 단절로 인한 충돌 현상의 발생 횟수
ConflictBound	ConflictCount가 가질 수 있는 최대 임계값
Tricklable	캐슁된 모든 화일의 write-back 지연 요구 가능 여부
TrickleCount	현재 서버로부터 write-back을 지연하도록 허가된 화일의 갯수

(a) DMF



(b) DCTLF

그림 2 DMF와 DCTLF의 엔트리 형태

본 논문의 제안 기법에서 요구하는 클라이언트의 데몬 프로세스들로는 TrickleDaemon 프로세스와 Client-Daemon 프로세스가 있다. 이들 중 TrickleDaemon 프로세스는 TrickleCount가 1이 되는 시점에 활성화되며 0이 되는 순간에 소멸한다. 활성화되어 있는 기간동안 TrickleDaemon 프로세스는 DMF를 관리하며, DMF 내에 기록된 화일들에 대해 각각 write-back이 지연된 시간을 계산하고 이 경과 시간을 TrickleInterval과 비교하여 write-back할 것인가를 결정한다. 또한, 단절된 후 재연결시 DMF내의 화일들에 대한 각 서버가 write-back이 지연되고 있음을 알고 있는지를 확인하여 필요한 경우 ClientDaemon에게 이를 알린다. Client-Daemon 프로세스는 클라이언트 내의 지역 캐쉬 관리 및 서버와의 통신, 그리고 TWB 기법의 전반적인 작업을 담당한다.

3.2.3 서버 측 데이타 구조

본 논문의 제안 기법을 위해서 서버 측에 요구되는 데이타 구조는 표 2 및 그림 3에서와 같다. 표 2에서는 서버 측에 필요한 각종 변수들을 보여주고 있다.

표 2 서버 측 데이타 구조

Delay-ConflictCount	저장된 각 화일에 대한 일정 기간내의 지연 충돌의 발생 횟수
Delay-ConflictMax	각 화일에 대한 지연 충돌의 최대 허용치

그림 3에서는 각 서버들간의 화일 일치성을 유지하기 위해 각 서버마다 갖는 FVA(File Version Array)의 각 엔트리 형식을 보인다.

본 논문의 제안 기법을 위해서 서버 측에는 ServerDaemon 프로세스가 존재한다. ServerDaemon 프로세스는 클라이언트와의 통신을 담당하며, Client-Daemon 프로세스와 함께 TWB 기법의 전반적인 작업을 담당한다.

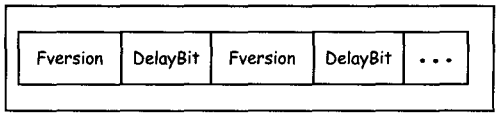


그림 3 FVA(File Version Array)의 각 엔트리 형태

3.3 제안 기법

본 논문에서 제안하는 TWB 기법이란 클라이언트 측에서 수정된 화일 중 자주 수정될 것으로 기대되는 화일에 대하여 서버로부터 write-back의 지연을 허가받아 해당 화일을 수정한 후 닫기와 동시에 화일을 바로 write-back 하지 않고 지연시키는 기법이다. 캐슁을 위해 우선 순위 기반의 알고리즘을 사용하며, 사용자에 의해 생성된 HoardDB 내의 우선 순위와 LRU 기법 기반의 우선 순위 조정 기법을 사용한다. 클라이언트는 우선 순위가 일정 수준 이상인 화일에 대해 서버에게 사용자의 개입 없이 자발적으로 write-back의 지연 허가를 요구하며, 허가된 화일들에 대해서는 주어진 시간 간격마다 write-back을 진행한다.

그림 4는 클라이언트와 서버내의 프로세스 사이의 동작 과정을 보인다. 클라이언트 내의 ClientDaemon 프로세스는 사용자로부터 open 요청을 받으면 자신의 지역 디스크 내의 캐쉬 정보를 확인하고 필요하면 서버로부터 화일을 접근하여 캐쉬할 수 있다. 또 ClientDaemon 프로세스는 사용자로부터 수정된 화일의 close 요청을 받으면 우선 순위를 계산한 후 서버로부터 write-back을 지연할 수 있도록 허락받는다. write-back 지연을 허락받으면, TrickleDaemon 프로세스를 활성화시킨다. TrickleDaemon 프로세스는 해당 화일에 대해 TrickleInterval을 확인하여, 주기적으로 write-back 하도록 ClientDaemon 프로세스에게 지시한다. 그림 4에서 작업 11과 작업 12는 TrickleInterval 동안 어떠한 수정 작업도 행해지지 않은 경우로 해당 화일에 대한 write-back 지연을 해제할 것을 서버에게 알리는 과정이다. 단절 상태가 발생한 경우에는 재연결시마다 TrickleDaemon 프로세스가 서버와 지연중인 화일들에 대해 화일 일치성을 확인하여 불일치가 발견되는 경우 ClientDaemon 프로세스에게 이를 알리는 과정이 이루어지는데 이는 작업 13과 14에서 보인다.

클라이언트에 의해 write-back이 지연중인 화일에 대해 또 다른 클라이언트로부터 접근이 발생되는 경우, 앞의 클라이언트는 write-back을 허용한 서버로부터 즉

시 write-back을 명령받을 수 있으며, 이 때 현재 최신의 데이타를 write-back한다. 이와 같은 상황을 지연 충돌이라 하며 서버 측에서는 지연 충돌이 잦은 화일에 대해 해당 화일에 대한 write-back 지연 허용을 금지시킬 수 있다.

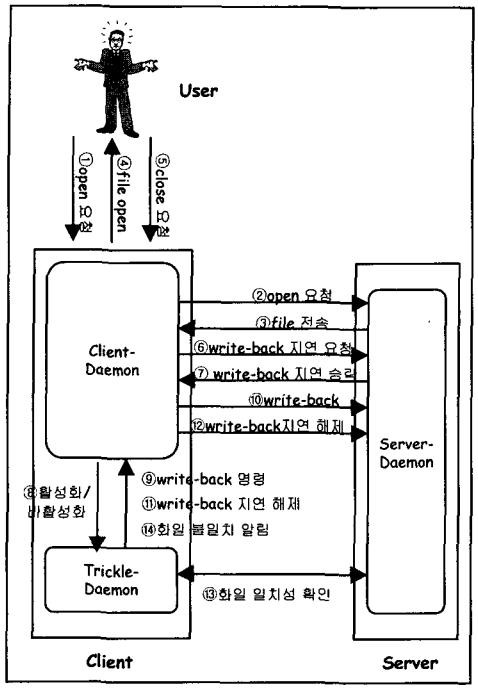


그림 4 본 제안기법의 전체적인 동작 과정

3.3.1 클라이언트

본 제안 기법에서 클라이언트 내의 전반적인 관리를 담당하는 ClientDaemon 프로세스의 알고리즘을 각 경우에 따라 알고리즘 1부터 알고리즘 5까지에서 보인다. 클라이언트의 입장에서 write-back을 지연중인 화일에 대해 write-back의 지연을 중지해야 할 경우에는 그 화일에 대한 DMF 내의 기록을 제거해야 하는데 이때의 작업절차는 알고리즘 3에서 보인다.

알고리즘 1에서는 클라이언트에서 특정 화일에 대한 열기(open) 요청이 발생한 경우의 처리 과정을 보여주고 있으며, 알고리즘 2에서는 열기후 수정된 화일에 대한 닫기(close) 요청이 있는 경우의 처리 과정을 보여주고 있다. ClientDaemon 프로세스는 사용자에 의해 수정된 화일에 대한 닫기 요청이 있는 경우, 해당 화일의 이름이 DMF 내에 존재하는지 확인하고, 존재하는 경우에는 화일의 갱신 관련 내용을 기록 화일에 추가로 기록한다. 해당 화일의 이름이 DMF 내에 존재하지 않는 경우에는 Tricklable의 값이 1인지 확인하고, 1인 경우에는 해당 화일에 대한 우선 순위를 계산하여 DelayBound보다 낮은 경우에는 바로 서버에 write-back 한다. ClientDaemon 프로세스에 의해 계산된 우선 순위가 DelayBound보다 큰 경우에는 서버에게 write-back을 지연시킬 수 있도록 요청하고, 이 요청이 수락되면 DMF 내에 해당 화일에 대한 정보를 등록한다. DMF 내에 해당 화일에 대한 내용을 등록할 때에는 화일 이름(Fname), 현재 시각(DelayStartTime)을 기록하고 ModifyBit을 1로 설정하며, 현재 write-back이 지연중인 화일의 갯수를 나타내는 TrickleCount를 증가시킨다. 만일 TrickleCount가 1인 경우에는 Trickle-Daemon을 활성화시켜 write-back이 지연중인 화일들을 관리하도록 한다. 클라이언트의 write-back 지연 요청이 거절되면 즉시 서버에게 write-back한다.

알고리즘 1 사용자에 의해 열기 요청이 발생한 경우

```
if (the file is not in the local cache
        or it is invalid)
    get the file from the server and cache it;
local_open the file;
```

알고리즘 2 사용자에 의해 수정된 화일의 닫기 요청이 발생한 경우

```
local_close the file;
if (the file has a DMF entry)
   log the status of the file into the log file;
else {
   if (Trickable equals 0)
      write_back the file to servers;
   else {
      compute priority of the file based on HoardDB
         contents and access history;
      if (priority less than DelayBound))
         write_back the file to servers;
      else {
         request to allow delayed_write_back
            to the server;
         wait(reply);
         if (delayed_write_back is rejected)
            write_back the file to servers;
         else {
            register the file in the DMF;
            increment TrickleCount;
            if (TrickleCount equals 1)
               activates TrickleDaemon;
            log the status of the file
               into the log file;
         }
      }
   }
}
```

지연 충돌이 발생한 경우 또는 write-back 지연중인 화일이 TrickleInterval 동안 어떠한 수정도 없을 때 더 이상의 write-back 지연을 해제하는 경우에는 DMF 내의 해당 화일에 대한 정보를 제거한 후, 서버에게로 해당 화일을 write-back해야 한다. DMF 내의 해당 화일에 대한 정보를 제거하는 과정은 알고리즘 3에서와 같다. 클라이언트 내의 임의의 프로세스에 의해 해당 화일이 사용중이라면, 그 프로세스가 사용하기 직전의 상태를 서버에게로 write-back해야 한다.

알고리즘 3 DMF 내의 특정 화일에 대한 기록을 제거하는 경우

```
remove the entry of the file in DMF;
decrement TrickleCount;
if (TrickleCount equals 0)
   deactivates TrickleDaemon;
```

특정 클라이언트에 캐싱되어 있는 화일에 대해 이 화일이 다른 클라이언트에서 수정되었음을 알리는 콜백 신호가 전달되는 경우에 ClientDaemon 프로세스는 자신이 관리하는 캐쉬 내의 해당 화일이 invalid 하다는 표시를 해 주게 된다.

클라이언트에서 지연중인 화일들 중 한 화일의 지연 시간이 TrickleInterval과 같아진 경우에, ClientDaemon 프로세스는 TrickleDaemon 프로세스로부터 서버로의 write-back 명령을 받게 되며, 이 때의 처리 과정을 알고리즘 4에서 보인다. 이 때 ClientDaemon 프로세스는 DMF 내의 해당 화일의 ModifyBit를 확인하여 그 값이 0인 경우에는 서버에게 해당 화일을 더이상 지연시키지 않겠다는 메시지를 보낸다. DMF 내의 해당 화일의 ModifyBit가 1인 경우에는 서버에게 해당 화일에 대한 최신의 정보를 write-back하고, 해당 화일에 대한 기록 화일 내의 모든 정보를 제거한 후 DMF 내의 ModifyBit을 0으로 설정한다. 이 때 만일 클라이언트 내의 임의의 프로세스에 의해 해당 화일이 사용중이라면 프로세스가 사용하기 직전의 상태를 write-back해야 한다.

알고리즘 4 TrickleDaemon 프로세스에 의해 서버로의 write-back 명령을 받는 경우

```
if (ModifyBit of the file in DMF equals 0) {
   request to release delayed_write_back
      to the server;
   remove the file information in DMF;
   // refer to algorithm 3
}
else {
   write_back the file to servers;
   remove all the transaction logging of the file;
   set DelayStartTime of the file in DMF
      to current time;
   set ModifyBit of the file in DMF to zero;
}
```

알고리즘 5에서는 단절된 상태에서 서버로부터 write-back 명령을 받았었다는 사실을 발견한 경우로, 이 때 ClientDaemon 프로세스는 해당 화일에 대한 기록 화일 내의 모든 작업을 시간의 역순으로 취소시키고 재실행하여야 한다. 또한 DCTLF(Disconnection Conflict Time Log File)에 현재 시간 t를 기록한 후, 일정기간(t - Δ) 동안의 충돌 횟수를 계산하여 ConflictCount에 기록한다. ConflictCount가 Conflict-Bound와 같아지는 경우에는 Tricklable의 값을 0으로 설정한다. 네트워크의 단절에 민감한 클라이언트가 TWB 기법의 적용을 받는 것은 오히려 더 큰 오버헤드를 초래할 수 있으므로, 클라이언트가 사용하는 모든 화일들에 대해 본 제안 기법을 적용 받지 않도록 하는 것이다.

알고리즘 5 TrickleDaemon 프로세스로부터 단절로 인한 충돌 발생 사실이 알려지는 경우

```
remove the file information from DMF ;
// refer to algorithm 3
undo all the operations executed to the file;
log current time in DCTLF
   and compute ConflictCount;
if (ConflictCount equals ConflictBound )
   set Trickleable to zero;
```

TrickleDaemon 프로세스에 관한 알고리즘은 알고리즘 6에서와 같다. TrickleDaemon 프로세스는 TrickleCount가 1 이상인 경우에 활성화된다. 활성화되어 있는 동안은 계속 DMF을 감시하며 각 화일의 DelayStartTime을 기준으로 하여 지연된 시간이 TrickleInterval과 같아지는 경우 ClientDaemon에게 서버로 해당 화일을 write-back 하도록 알려준다. 또한, 알고리즘 7에서와 같이, 단절된 후 재연결되는 즉시 DMF 내의 화일들 각각에 대해 서버와의 일치성을 확인하여야 한다. 만일 충돌이 발생하면 ClientDaemon 프로세스에게 단절 상태에서 서버에게 write-back 명령을 받았다는 사실을 알려준다.

알고리즘 6 TrickleDaemon의 상시 처리 과정

```
while(1) {
   for (each entry in DMF) {
      sets  delayed_time
         to (current_time - DelayStartTime);
      compare TrickleInterval with delayed_time;
      if (TrickleInterval equals delayed_time);
         inform ClinetDaemon process
            to write_back the file to servers;
   }
}
```

알고리즘 7 재연결 시점에서의 TrickleDaemon의 처리 과정

```
check the consistency of all the files in DMF;
for (each inconsist file)
    inform ClinetDaemon process of the fact that
       he had received write-back command from
       server during disconnection state;
```

3.3.2 서버

본 제안 기법에서 서버 내의 전반적인 관리를 담당하는 ServerDaemon 프로세스는 FVA(File Version Array) 내의 화일의 수정 버전(Fversion)과 함께, 관련된 클라이언트 중에 현재 write-back 지연을 허용 받은 클라이언트가 있는지의 여부(DelayBit)에 의해 관련된 다른 서버들간의 화일 일치성을 유지시킨다. 임의의 클라이언트로부터 수정된 화일이 write-back되는 경우에는 해당 화일을 저장하고 FVA의 화일 버전을 증가시킨다. 서버가 임의의 클라이언트로부터 화일에 대한 읽기 요청을 받은 경우에 ServerDaemon 프로세스의 처리 과정을 알고리즘 8에서 보이며, write-back의 지연 요청을 받은 경우에 대한 처리 과정은 알고리즘 9에서 보인다.

알고리즘 8에서는 클라이언트로부터 화일 읽기 요청을 받은 경우에 대한 서버의 처리 과정을 보인다. ServerDaemon 프로세스는 해당 화일에 대한 FVA 내의 DelayBit들 중 한 비트가 1이라면 지연 충돌이 발생한 경우이므로, 해당 화일의 DelayConflictCount를 증가시킨다. 현재시간(t)에 대해 일정기간(t - δ) 동안의 DelayConflictCount가 DelayConflictMax와 같아지면 해당 화일이 둘 이상의 클라이언트에 의해 동시에 자주 접근되는 화일이라고 판단하여 그 화일에 대한 FVA의 모든 DelayBit들을 1로 설정한다. 이렇게 함으로써 해당 화일을 참조하는 모든 클라이언트들에게 해당 화일에 대해 더이상의 지연을 허용하지 않도록 한다. 이 때 write-back을 지연하도록 허용한 서버는 지연중인 클라이언트에게 바로 write-back을 명령해야 하며, 만일 해당 클라이언트와 단절된 상태라면 바로 FVA의 DelayBit에 0을 설정하고 다른 서버들에게 FVA를 보낸 후, 화일을 요구한 클라이언트에게 해당 화일을 전송하도록 한다.

알고리즘 8 클라이언트로부터 화일 읽기 요청을 받은 경우

```
if (one of the DelayBits in FVA
    of the file equals 1)
{
  increment DelayConflictCount;
  if (DelayConflictCount equals
        DelayConflictMax during t-δ)
     set all DelayBits in FVA to 1;
  if (there exists other server that has
        allowed delayed_write_back)
     send write_back_command message to the server;
  else {
     request write_back to the client;
     if (disconnected with the client) {
        sets DelayBit to 0;
        send FVA to other servers;
     }
  }
}
send the file to the client;
```

알고리즘 9에서는 서버가 클라이언트로부터 write-back 지연 요청을 받은 경우에 대한 알고리즘을 보인다. 이 경우 ServerDaemon 프로세스는 해당 화일에 대한 FVA를 확인하여, 모든 DelayBit들이 1인 경우에는 바로 write-back 하도록 한다. 그렇지 않은 경우에는 자신의 DelayBit에 1을 설정하고 관련된 다른 모든 서버들에게 write-back 지연을 허용한다는 사실을 알린 후, 요청한 클라이언트에게 write-back 지연을 허용한다. 해당 화일을 관리하는 모든 서버들은 관련된 모든 클라이언트들에게 화일이 수정되었음을 알려주어야 한다.

알고리즘 9 클라이언트로부터 write-back 지연 요청을 받은 경우

```
if (all DelayBits within FVA equal 1)
   reject the request;
else {
   sets DelayBit in FVA to 1;
   sends the file's FVA to other servers;
   allows the client to delay write_back;
}
```

4. 시뮬레이션 및 평가

본 논문에서는 시뮬레이션 도구로 Devs-scheme[17]을 사용한다. Devs-scheme은 이산 사건 시스템(discrete event system)의 시뮬레이션에 적합한 것으로 알려져 있다. 4.1 절에서는 시뮬레이션을 위한 몇 가지 가정 및 시뮬레이션 환경에 대해 기술하고, 4.2 절에서는 시뮬레이션에 의한 분석 결과를 보인다.

4.1 시뮬레이션 환경

본 논문의 시뮬레이션을 위해 이산 사건 시스템에 잘 적용되는 Devs-scheme을 이용했다. 시뮬레이션에 의해 사용자로부터 수정이 잦은 임의의 화일에 대해 관찰하며, 단위 시간은 1로 하고, 전체 관찰 기간을 20160으로 하였다. 화일이 닫힌 후 다시 열릴 때까지의 시간 간격(close to open time interval)과 화일이 사용되는 시간, 즉, 화일이 열린 후 닫힐 때까지의 시간 간격(open to close time interval)은 지수 분포(exponential distribution)를 따른다고 가정한다. 또한 임의의 클라이언트에 의해 write-back이 지연되고 있는 화일에 대해 또 다른 클라이언트로부터 읽기 요청이 발생(지연 충돌)하는 시간 간격 역시 지수 분포를 따른다고 가정한다. 클라이언트는 항상 Tricklable한 상태라 가정한다.

4.2 시뮬레이션 측정 결과

본 논문에서 제안한 기법의 시뮬레이션을 위하여 COTI(close to open time interval)와 OCTI(open to close time interval) 개념을 사용한다. 이 개념은 한 사용자가 평균 화일을 여는 시간과 한 번 연상태에서 다시 그 화일을 닫는데 걸리는 평균 시간으로 화일들의 사용빈도를 구분하고자 제안한 개념이다. 이 개념으로 각 화일에 대한 사용 빈도를 화일이 닫힌 후 다시 열릴 때까지의 시간 간격의 평균 COTI와 화일이 사용되는 시간, 즉, 화일이 열린 후 닫힐 때까지의 시간 간격의 평균 OCTI를 기준으로 하여 다음과 같이 분류하여 총 12 가지의 경우에 대해 성능을 측정하였다.

▷ COTI : 180, 360, 540
▷ OCTI : 30, 60, 90, 120

클라이언트로부터 서버에게 전송되는 메시지의 총 전송 횟수(T)는 Tregular, Ttrickle, Tconflict 그리고 Tsignal의 합으로 이루어지며, 그 각각은 표 3에서와 같이 정의한다.

그림 5에서는 총 관찰 기간을 20160으로 하고, TrickleInterval을 1440으로 설정하여 지연충돌 발생 간격의 평균이 각각 720, 1440, 7200, 10080일 때의 사용빈도와 총 전송 횟수의 관계에 대한 시뮬레이션 결과를 보인다. 여기에서 DC는 지연충돌(delay conflict)을 나타낸다.

표 3 클라이언트-서버간 총 전송 횟수를 구성하는 요소

T = Tregular + Ttrickle + Tconflict + Tsignal	
Tregular	제안 기법이 적용되지 않고 바로 write-back 되는 횟수
Ttrickle	제안 기법이 적용된 경우로, TrickleInterval이 되어 write-back되는 횟수
Tconflict	제안 기법이 적용된 경우로, 지연 충돌이 발생하여 강제적으로 write-back 되는 횟수
Tsignal	제안 기법이 적용된 경우로, TrickleInterval이 되었을 때 수정되지 않은 상태이므로 write-back 지연 상태를 종료하는 메시지를 서버로 전송하는 횟수

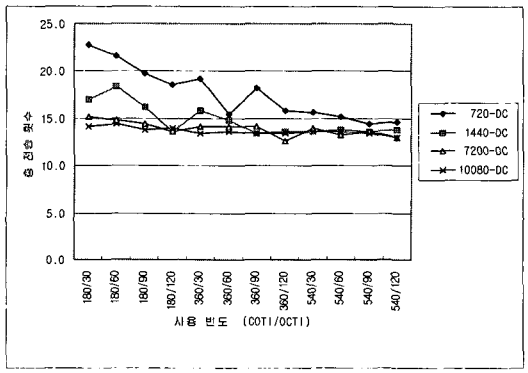


그림 5 지연충돌 발생 간격별 사용 빈도와 총 전송 횟수의 관계

그림 5에서의 720-DC의 경우에는 평균 720 단위 시간마다 지연충돌이 발생될 것으로 기대되는 화일에 대한 사용 빈도와 총 전송 횟수의 관계를 나타낸다. 이 경우에서 화일의 사용 빈도가 높은 경우(COTI/OCTI의 값이 180/30의 경우)의 약 23회의 총 전송 횟수에 비해 사용 빈도가 낮은 경우(COTI/OCTI의 값이 540/120의 경우)에는 총 전송 횟수가 약 15회로 34.78% 감소되었다. 동일한 사용 빈도로 COTI/OCTI의 값이 180/30의

경우에 대해서는, 720-DC의 경우에 약 23회의 총 전송 횟수에 비해 10080-DC의 경우에서 총 전송 횟수가 약 14회로 39.13% 감소되었으며, COTI/OCTI의 값이 540/120의 경우에 대해서는, 720-DC의 경우의 약 15회의 총 전송 횟수에 비해 10080-DC의 경우에서는 총 전송 횟수가 13회로 13.33% 감소되었다.

따라서, 사용빈도가 높은 화일의 경우(COTI/OCTI의 값이 180/30의 경우)가 사용빈도가 낮은 화일(COTI/OCTI의 값이 540/120의 경우)에 비해 지연충돌의 발생 간격에 더 민감함을 알 수 있다. 지연충돌이 드문 화일에 대해 본 논문의 제안 기법이 좋은 성능을 보인다.

그림 6에서는 그림 5에서 보이고 있는 제안 기법을 적용한 경우와 CFS 등에서와 같이 가장 일반적으로 사용되는 수정된 화일을 닫을 때마다 write-back 하는 경우(기존 기법)와의 비교를 보인다. 화일의 사용 빈도가 높을수록 본 논문의 제안 기법의 성능이 우수함을 볼 수 있다.

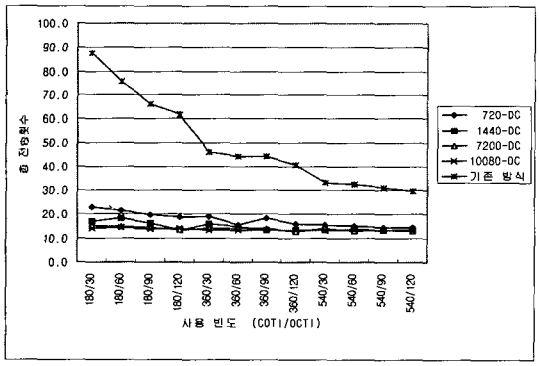


그림 6 지연충돌 발생 간격별 사용 빈도와 총 전송 횟수와의 관계(기존 기법과 제안 기법의 비교)

그림 7과 그림 8에서는 총 관찰 시간을 20160으로 하고, 지연충돌 발생 간격을 10080으로 가정한 후, TrickleInterval이 각각 360, 720, 1440, 2160, 2880, 3600일 때의 사용 빈도와 총 전송 횟수의 관계에 대한 시뮬레이션 결과를 보인다. 총 관찰 시간동안 지연충돌은 평균 2번 정도 발생할 가능성이 있다고 가정한 것이다. 그림 7에서는 각 TrickleInterval에 대한 사용 빈도에 따른 총 전송 횟수를 비교한다.

그림 7에서 360 T_time의 경우에는 TrickleInterval을 360으로 설정한 경우를 나타낸다. 이 경우에서 화일의 사용 빈도가 높은 경우(COTI/OCTI의 값이 180/30인 경우)의 약 50회의 총 전송 횟수에 비해 사용 빈도가 낮은 경우(COTI/OCTI의 값이 540/120인 경우)에는 약 33회의 총 전송횟수로 34% 감소되었다. 동일한 사용 빈도로 COTI/OCTI의 값이 180/30인 경우에 대해서는 360 T_time의 경우의 약 50회의 총 전송 횟수에 비해 3600 T_time의 경우에는 약 7회의 총 전송 횟수로 86%의 큰 감소가 발생하였다. 따라서 총 전송 횟수는 TrickleInterval Time에 매우 민감함을 알 수 있다.

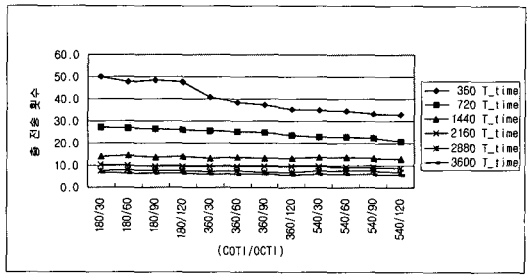


그림 7 TrickleInterval별 사용 빈도에 따른 총 전송 횟수

그림 8에서는 그림 7에서 보이고 있는 제안 기법을 적용한 경우와 CFS 등에서와 같이 가장 일반적으로 사용되는 수정된 화일을 닫을 때마다 write-back 하는 경우(기존 기법)와의 비교를 보인다. 그림 8에서 TrickleInterval을 360으로 설정한 경우에는 COTI/OCTI의 값이 540/30인 경우보다 사용 빈도가 낮은 경우에 본 논문의 제안기법을 적용한 경우가 기존의 방식을 그대로 적용한 경우보다도 성능이 낮아짐을 볼 수 있다. TrickleInterval을 너무 짧게 설정하면 성능향상을 기대하기가 어렵고, TrickleInterval을 너무 길게 설정하면 지연 충돌 발생 확률이 높아질 것이다. 따라서 TrickleInterval을 적절히 설정했을 때, 사용 빈도가 높

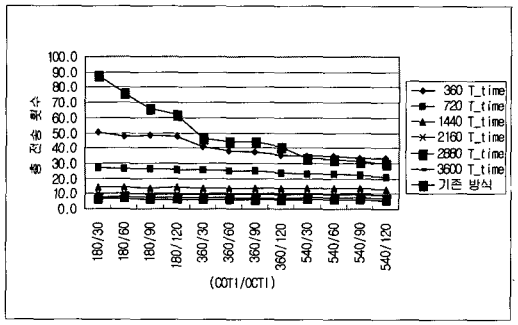


그림 8 TrickleInterval별 사용 빈도에 따른 총 전송 횟수와 기존 기법과의 비교

은 화일에 대하여 제안 기법의 성능이 높다는 것을 알 수 있다.

TrickleInterval을 크게 하면, 총 전송량을 줄일 수 있어 대역폭을 절약할 수 있는 반면, 지연충돌의 발생 빈도가 높아질 수 있다. 이동 컴퓨팅 환경에서는 완전히 단절된 상태보다는 약한 연결 상태가 많이 나타나므로 [16], 지연충돌이 발생하면 서버에 의해 write-back 명령을 받아 처리할 수 있다. 그러므로 이동 컴퓨팅 환경에서의 지연충돌은 치명적인 오버헤드를 발생시키지는 않는다. 그러나 단절시의 지연충돌은 역시 시스템의 성능 저하를 초래하게 될 것이므로, TrickleInterval과 지연 충돌 발생 간격의 적절한 분석과 고려가 요구된다.

5. 결 과

본 논문의 제안 기법은 현재 Unix 환경에서 공유된 화일 중 대부분이 실제로 동시에 여러 사용자로부터 쓰기 공유되는 경우가 드물다는 연구 보고에 근거를 둔다 [6, 7, 8, 9]. 또한 이동 컴퓨팅 환경이 더욱 일반화되고, 이에 따라 많은 화일들이 공유의 대상이 되는 환경에서는 총 공유 화일의 수에 비해 여러 사용자에 의해 동시 접근되는 화일의 수가 상대적으로 훨씬 적어질 것이라 기대되므로 본 제안 기법이 더 좋은 성능을 낼 수 있을 것이다.

본 제안 기법은 기존에 제안된 write-back 지연 기법들과는 달리 약한 연결성을 이용한 기법이다. 본 기법에서는 Trickle Write-Back의 개념을 사용함으로써 write-back을 지연시키는 효과뿐만 아니라 기록 화일로 인한 디스크 낭비와 단절 상태로 인한 쓰기 충돌시의 손실을 최소화하는 이점을 얻을 수 있다. 또한 임의의 클라이언트가 write-back을 지연시키는 동안에 또 다른 클라이언트로부터 동일 화일에 대한 접근이 요구되는 경우에는 서버가 지연중인 클라이언트에게 write-back을 명령할 수 있도록 함으로써 write-back을 지연시킴으로 인하여 추가로 발생할 수 있는 읽기/쓰기 충돌을 방지한다.

서버는 둘 이상의 사용자로부터 동시 접근이 자주 발생하는 화일에 대해서 더 이상의 write-back 지연을 허용하지 않도록 할 수 있고, 임의의 클라이언트의 잦은 단절로 인해 쓰기 충돌이 자주 발생되는 경우, 클라이언트 자체적으로 write-back의 지연을 요구하지 않도록 했다. 결과적으로 단절 상태가 잦은 클라이언트나 쓰기 충돌이 잦은 화일에 대해 고려함으로써 치명적인 오버헤드를 해결한 것이다.

향후 이동 컴퓨팅 환경의 규모가 커지고 사용자가 급속히 증가할수록 대역폭의 제약에 대한 문제는 더욱더 심각해 질 것이며, 이에 대한 해결 기법이 보다 활발히 연구되어져야 할 것이다. 또한 앞으로 TrickleInterval을 네트워크의 상태에 따라 동적으로 할당할 수 있도록 하는 부분에 대해서도 연구가 필요할 것으로 보인다.

참 고 문 헌

[1] Choi, E., "Network Design in the Era of Internet," In Proc. SWCC'98, 한국정보과학회, Aug. 1998.

[2] Baker. M., Hartman, J., Kupfer, M., Shirriff, K., and Ousterhout, J., "Measurements of a Distributed File System," In Proc. Thirteenth ACM Symposium on Operating System Principles, Pacific Grove, CA., Oct. 1991.

[3] Chen, K., "Write Caching in Distributed File Systems," Master's thesis, University of Saskatchewan, Saskatoon, SK., 1994.

[4] Froese, K., "File System Support for Weakly Connected Operation," In Proc. Seventh Annual Graduate Symposium on Computer Science, Saskatoon, SK., Mar. 1995.

[5] Mann, T., Birrell, A., Hisgen, A., Jerian, C., and Swart, G., "A Coherent Distributed File Cache with Directory Write-behind," SRC Research Report 103, Digital Equipment Corporation, Palo Alto, 1993.

[6] Kistler, J., and Satyanarayanan, M., "Disconnected Operation in the Coda File System," ACM Transactions on Computer Systems Vol. 10, No. 1, Jan. 1992.

[7] Kuenning, G., et. al., "An Analysis of Trace Data for Predictive File Caching in Mobile Computing," In Proc. 1994 Summer USENIX Conference, Los Angeles, CA., Jun. 1994.

[8] Satyanarayanan, M., Kistler, J., Mummert, L., Ebling, M., Kumar, P., and Lu, Q., "Experience with Disconnected Operation in a Mobile Computing Environment," In Proc. First USENIX Symposium on Mobile and Location-Independent Computing, Cambridge, MA., Apr. 1993.

[9] Birell, D., Hisgen, A., Jerian, C., Mann, T., and Swart, G., "The Echo Distributed File System," SRC Research Report 111, Digital Equipment Corporation, Palo Alto, 1993.

[10] Coulouris, G., Dollimore, J., and Kindberg, T., Distributed Systems, Addison-Wesley, NY., 1993.

[11] Mullender, S., Distributed Systems,ACM Press, NY., 1994.

[12] Mummert, L., Ebling, M., and Satyanarayanan, M., "Exploiting Weak Connectivity for Mobile File Access," school of Computer Science, Carnegie Mellon University, 1995.

[13] Froese, K., and Bunt, R., "Issues in File Cache Management for Mobile Computing," Dept. of Computer Science, University of Sask., Canana, Jul. 1996.
[14] Mummert, L., and Satyanarayanan, M., "Large Granularity Cache Coherence for Intermittent Connectivity," Carnegie Mellon University.
[15] Froese, K., "File Cache Management for Mobile Computing," M. Sc. Thesis Proposal, Supervisor Prof. Richard Bunt, Dept. of Computer Science, University of Sask.
[16] Imielinski, T., and Badrinath, B., "Mobile Wireless *Computing : Challenges in Data Management*," Technical Report, Dept. of Computer Science, University of Rutgers, Oct. 1994.
[17] Bernard P., Object-Oriented Simulation with Hierarchical, Modular Models, Academic Press, 1990.

김 문 정

1998년 2월 성균관대학교 정보공학과 졸업(학사). 1998년 3월 ~ 현재 성균관대학교 전기전자 및 컴퓨터공학부 석사과정. 관심분야는 이동 컴퓨팅, 분산 시스템

엄 영 익

1983년 서울대학교 계산통계학과 졸업(학사). 1985년 서울대학교 대학원 전산과학전공(석사). 1991년 서울대학교 대학원 전산과학전공(박사). 1983년 ~ 1986년 서울대학교 도서관 전산화준비실. 1986년 ~ 1993년 단국대학교 전자계산학과 부교수. 1993년 ~ 현재 성균관대학교 전기전자 및 컴퓨터공학부 교수. 관심분야는 분산 시스템, 이동 컴퓨팅, 분산 객체 시스템, 운영체제