스트리밍 미디어 캐싱을 위한 사용자 수준 화일 시스템

(A User-Level File System for Streaming Media Caching)

오 재 학 † 차 호 정 ††

(Jaehak Oh) (Hojung Cha)

요 약 본 논문에서는 스트리밍 미디어의 효율적인 캐싱과 재전송을 목적으로 범용 화일 시스템에 기반한 사용자 수준의 스트리밍 미디어 캐쉬 화일 시스템(umcFS)을 설계하고 구현하였다. umcFS는 범용 화일 시스템의 저장역을 선점하는 화일 디스크에 기반한다. 화일 디스크는 캐쉬 블럭의 물리적인 연속성을 확보하여 효율적인 캐쉬 입출력 시스템을 구성하는 가상 디스크이다. umcFS 구조는 큰 캐쉬 블럭 구조와 제어블럭의 정적 할당을 기본 정책으로 유닉스 화일 시스템과 비교되는 확장된 1차 간접 블럭 참조를 통해서 캐쉬 블럭을 관리한다. 또한 사용자 수준의 라이브러리로 개발되어 시스템 간에 이식성과 확장성이 우수하고 개발 기간이 짧은 장점이 있다. umcFS의 구현을 통해 umcFS의 저수준과 API 수준의 입출력 성능을 비교 분석하였다. 1024KB의 적정 캐쉬 블럭 크기에 대한 임의 입출력 실험 결과, umcFS가 화일 블럭 캐쉬에 비해 약 13%의 성능 향상을 보임을 알 수 있었다.

키워드 : 미디어 캐싱, 화일 시스템

Abstract This paper presents the design and implementation of a cache file system, umcFS, which is specifically designed to provide an efficient caching and transmission of streaming media. The proposed file system is based on the concept of file disk and implemented as an application level on top of a general-purpose file system. The file disk favors the continuity of cached media and provides an efficient I/O mechanism for cache server. umcFS statically allocates control blocks as well as media cache blocks. These blocks are referenced by the single-level indirect management structure. As the file system is designed as an application level, it is easy to develop and port to other systems. The performance of the implemented system shows that umcFS performs about 13% better than the native file system in randomly accessing the cache blocks of 1024KB.

Key words : media caching, file system

1. 서 론

최근에 미디어 스트리밍 분야는 서비스의 유료화와 콘텐츠 품질의 향상을 위해 스트리밍 미디어 캐싱 기술의 개발과 캐싱 네트워크의 구성에 많은 관심을 보이고 있다. 스트리밍 미디어 캐싱은 지역 네트워크 캐싱 시스템을 기반으로 사용자가 선호하는 콘텐츠의 일부를 저장하고 관리하는 기술이다. 사용자의 요구가 발생했을 때 캐싱된 콘텐츠와 일치하면, 일부의 콘텐츠를 지역 네트워크의 캐싱 시스템에서 전송받아 사용자 대기 시간과 원격 스트리밍 서버의 콘텐츠 전송량을 감소시키는 효과가 있다. 또한 기간 네트워크 트래픽을 지역망으로 이전시키는 효과가 있어 전체 네트워크 트랙픽의 균일화를 유도한다[1]. 스트리밍 미디어 서비스가 일반화된 사용자 서비스로써 자리잡은 반면에 기존의 시스템들은 일대일 전송 서비스 모델을 가정하고 있으며, 스트리밍 미디어의 형식과 프로토콜이 상이한 현실은 스트리밍 미디어 캐싱 시스템의 개발에 제한사항이 되고 있다[2].

스트리밍 미디어 캐싱 시스템의 핵심요소는 캐쉬 입출력 시스템이다. 스트리밍 미디어의 캐쉬 입출력은 서

· 본 연구는 정보통신부 대학기초 연구지원 사업(과제번호 : 2001-076-3) 지원에 의해 연구되었음.

† 학생회원 : 광운대학교 컴퓨터과학과
ojh@cs.kwangwoon.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
hjcha@cs.yonsei.ac.kr

논문접수 : 2001년 10월 23일
심사완료 : 2002년 6월 18일

버로부터 콘텐츠 입력과 클라이언트로 출력하는 양방향 구조로 스트리밍 서버와 비교하여 최대 두배의 입출력량을 갖는다. 따라서 효율적인 입출력 시스템의 설계는 스트리밍 미디어 캐싱 시스템의 성능과 밀접한 관계에 있다. 캐쉬 입출력 시스템은 캐쉬의 저장 미디어와 입출력 방식에 따라 설계되어야 한다. 캐쉬의 저장 미디어는 원본 미디어와 패킷 미디어로 구분할 수 있다. 원본 미디어는 서버로부터 콘텐츠를 복사하거나 스트리밍프로토콜에 의해 전송받은 패킷을 복원한 것이다. 원본 미디어를 이용한 입출력 시스템은 서버의 입출력 시스템과 동일하게 구성하거나 미디어와 캐쉬의 특성을 반영한 입출력 시스템을 새로 구성해야 한다. 반면에 패킷 미디어는 스트리밍 프로토콜 수준에서 전송된 패킷이 저장 단위이고 패킷들의 일정량을 캐쉬의 입출력 단위로 구성한다. 패킷 미디어 방식은 미디어 재구성이 필요없고 프로토콜 정보를 입출력 정보로 재구성할 필요가 있다.

입출력 방식은 범용 화일 시스템 상의 화일 입출력과 전용 캐쉬 화일 시스템의 구축을 고려 할 수 있다. 범용 화일 시스템 상에서 화일 단위의 입출력 구조는 화일과 디렉토리를 이용해 쉽게 구현할 수 있으나 여러 단점들이 있다. 캐싱된 미디어의 선호도에 따라 캐싱량의 조절시 대상 미디어를 재구성하는 입출력 부하가 발생하며 화일 블럭 단위의 캐쉬 구조의 설계시 미디어의 대용량성에 비례해 화일의 수가 증가하므로 화일 시스템의 운영 부하를 가중시킨다. 또한 화일 시스템 제어를 위한 open/close 시스템 콜을 빈번하게 발생시키고 특성화된 멀티미디어 화일 시스템에 이용시 입출력 보장 요구를 적용할 수 없거나 적용하더라도 성능을 낮추는 결과를 초래한다. 한편 전용 캐쉬 화일 시스템의 구성은 커널과 사용자 수준의 개발로 나눈다. 커널 수준의 개발은 저수준 입출력을 직접 관리함으로서 효율성을 극대화 할 수 있는 장점이 있지만, 개발기간이 길고 시스템 이식성이 낮은 단점이 있다. 사용자 수준의 개발은 본 논문에서 제안하는 방식으로 범용 화일 시스템을 기반으로 개발기간을 단축할 수 있고, 이식성이 뛰어난 장점이 있다. 또한 멀티미디어 화일 시스템에 기반할 경우 저수준의 입출력을 제어할 수 있는 장점이 있다.

본 논문과 관련된 연구에는 멀티미디어 화일 시스템과 버퍼 캐쉬 분야가 있다. 멀티미디어 화일 시스템은 대용량 화일 시스템 지원, 입출력 보장 정책, 빠른 복구 정책, 실시간 입출력 스케쥴링 등을 연구되어 왔고 멀티미디어 데이타 편집과 스트리밍 서버의 입출력 시스템으로 활용되고 있다[3][4]. 최근에 연구되고 있는 MMFS[5]는 기존에 멀티미디어 화일 시스템과 구별되는 이차원 화일 구조를 제안한다. 이차원 화일 구조는 멀티미디어에 존재하는 다중화 구조와 유사한 개념으로 유닉스의 일차원 화일 구조에 독립적으로 제어 가능한 여러 미디어를 위치시킬 수 있다. 이차원 화일을 MM 화일이라고하며 하부의 미디어는 strand라 한다. MMFS의 MM 화일과 strand 구조는 멀티미디어 편집과 복합 등에 편리한 구조이며 API 수준에서 미디어 정보를 얻을 수 있어 멀티미디어 프로그램 개발에 유용한 환경을 제공한다. MMFS는 스트리밍 서버의 편도 데이타 전송과 멀티미디어 편집 프로그램의 대용량 입출력 경향을 잘 반영한다. 반면에 캐쉬 입출력은 고정 크기 캐쉬 블럭에 대한 양방향 입출력 구조이고 캐쉬 블럭 구성을 위해 화일 블럭 구조를 활용해야 한다. MMFS에서 이차원 화일 구조는 캐쉬 블럭 구성에 적절하지 않으며, 캐쉬 구성은 open/close 시스템 콜 수행에 따른 자원 선점과 해제를 반복하는 비효율적인 구조가 될 수 있다. 버퍼 캐쉬 연구는 디스크나 네트워크로부터 버퍼로 전송된 미디어의 일부분을 동일한 요구에 대해서 효율적으로 재활용할 수 있는 방법을 제안한다. 버퍼 캐쉬 기술은 서버로부터 서비스 부하를 줄이고 클라이언트의 초기 지연 시간을 줄이는 효과가 있다. 버퍼 캐쉬는 서버와 클라이언트 풀 모델로 구분한다. 서버의 버퍼 캐쉬 구성은 서비스 중인 동일한 미디어 요구에 대해 선행 요구로 인해 디스크로부터 버퍼로 읽어들인 미디어의 일부가 후행 요구의 유효 범위 내에 있다면 재활용하는 기술이다. 대표적인 예로 간격(interval) 캐싱이 있다[6]. 클라이언트의 버퍼 캐쉬 풀(pool) 구성은 선택한 미디어를 서버에 요구하기 전에 이웃한 클라이언트에 관련 미디어가 있는지 확인하는 정책이다. 유효 범위 내에 있는 미디어가 이웃한 클라이언트에 존재하는 경우 서버를 거치지 않고 일부 혹은 전체를 빠르게 전송 받는다. 두 모델에서 처럼 버퍼에 캐싱된 미디어가 버퍼 정책에 따른 적절한 간격 결정과 요구된 미디어를 가진 가까운 호스트를 결정하는 문제는 쉽지가 않고, 캐쉬 입출력 시스템과 비교해 전송의 효율성에 중점을 두고 있다.

본 논문에서는 스트리밍 미디어의 효율적인 캐싱과 재전송을 목적으로 범용 화일 시스템에 기반한 사용자 수준의 스트리밍 미디어 캐쉬 화일 시스템(umcFS)의 설계와 구현을 기술한다. umcFS는 작은 크기의 패킷 미디어들를 캐쉬 블럭 단위로 묶어 디스크 기반에 전형적인 캐쉬 입출력 시스템을 유형화 한다. 화일 디스크 구성은 범용 화일 시스템 위에 물리적인 데이타 블럭의 연속성을 최대한 확보하여 논리적인 캐쉬 저장 공간을 제공하고 계층적 캐쉬 구성을 위한 확장성을 제공한다.

umcFS 구조는 큰 캐쉬 블럭 구조와 제어블럭의 정적 할당을 기본 정책으로 유닉스 화일 시스템과 비교되는 확장된 1차의 간접 블럭 참조를 통해서 캐쉬 블럭을 관리하고 사용자 수준의 라이브러리로 개발되어 시스템 간에 이식성과 확장성, 단기 개발 등이 고려되었다.

논문의 구성은 다음과 같다. 2장에서 스트리밍 미디어 캐슁 화일 시스템의 세부사항을 기술한다. 3장에서는 캐쉬 블럭을 화일 단위로 관리하는 화일 블럭 캐쉬와 스트리밍 미디어 캐슁 시스템의 개발을 통해 umcFS의 저수준과 API 수준의 입출력 성능을 비교 검증하고 4장에서 결론으로 맺는다.

2. umcFS 캐쉬 화일 시스템

umcFS는 스트리밍 미디어 캐슁 시스템에서 캐쉬의 입출력을 제어하는 사용자 수준의 캐쉬 화일 시스템이다. umcFS는 스트리밍 미디어의 효율적인 저장과 재전송 구조로 설계되었으며 범용 화일 시스템에 기반한 사용자 수준의 입출력 제어를 지원한다.

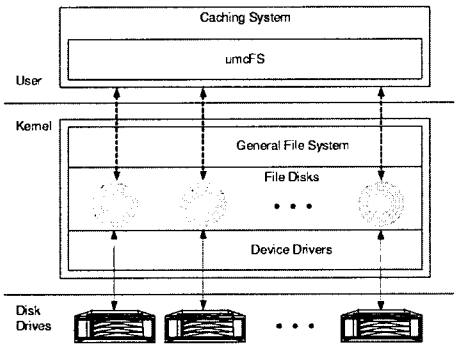


그림 1 umcFS 개요

2.1 umcFS 개요

그림 1은 스트리밍 미디어와 범용 화일 시스템과의 상호 관계를 통해 umcFS 개요를 보여준다. umcFS의 특징은 다음과 같다. 첫째, umcFS는 스트리밍 미디어에 기반한 캐쉬 입출력 구조이다. 스트리밍 미디어는 표준 멀티미디어 프로토콜의 실시간 특성을 반영한 작은 패킷으로 전송된다. 네트워크와 캐쉬의 입출력이 작은 데이타를 단위로 처리하게 되면 빈번한 시스템 콜의 호출로 비효율적인 구조가 된다. umcFS에서는 캐쉬 입출력 처리 부하를 줄이기 위해 umcFS의 블럭과 네트워크 입출력 버퍼의 크기를 동일하게 구성하여 입출력 단위로 구성하였다. 또한 다중 패킷을 구성할 수 있는 블럭 구조와 실험에 의한 효율적인 블럭 크기를 제안하여 시스템 콜을 최소화하였다. 둘째, umcFS는 범용 화일 시스템에 기반한 사용자 수준의 입출력 제어 구조이다. 커널 수준의 캐쉬 화일 시스템의 개발은 효율적인 입출력 구성에 좋은 모델이 될 수 있으나, 개발기간이 길고 캐슁 시스템의 범용성과 이식성이 약한 단점이 있다. umcFS는 사용자 수준의 라이브러리 구조로 설계되어 이식성과 범용성이 좋으며 범용 화일 시스템이 지원하는 64-bit 주소 범위와 Direct I/O 등의 유용한 기능을 이용한 효율적인 입출력 체계를 지원한다. 또한 라이브러리 구조의 umcFS API를 구성함으로써 캐슁 시스템의 독립적인 개발을 지원한다. 셋째, umcFS는 입출력 제어를 위해 화일 디스크라는 가상 디스크를 구성한다. 화일 디스크는 디스크 드라이브에 화일 시스템을 구성하듯이 범용 화일 시스템을 기반으로 선형적으로 구성된 가상 디스크이며 umcFS의 제어 구조를 통해 캐쉬 입출력을 수행한다. 넷째, umcFS는 유닉스 화일 시스템과 유사한 블럭 관리 구조이며 1차 간접 블럭 참조 구조이다. 일반적으로 유닉스 화일 시스템의 블럭 구조는 다단계 참조를 통해 목표로 하는 데이타를 입출력한다. 유닉스 화일 시스템은 다단계 참조에 따른 화일 시스템의 자체 부하를 가지고 있어 화일의 연속성 확보에 취약한 구조이다. umcFS는 1차의 간접 참조의 단순 구조와 큰 블럭 구조를 지원하여 시스템의 처리량을 줄이고 블록 단위의 입출력 연속성을 보장한다. 다섯째, umcFS의 블럭은 스트리밍 미디어의 전송 패킷을 관리하기 위한 캐슁 단위이다. umcFS의 블럭 구조는 블럭 내의 패킷 정보를 담는 블럭 헤더와 패킷들로 구성된다. 블럭 헤더는 패킷의 시간과 순서 값에 기반한 제어 구조를 구성하여 프로토콜 정보의 검색과 패킷의 입출력을 용이하게 하며 전송 패킷의 캐슁을 위한 일관성있고 효율적인 구조를 제공한다. 여섯째, umcFS는 유닉스 화일 시스템의 블럭 참조 구조와 버퍼 캐쉬 구조를 착안하여 설계되었고 사용자 수준의 입출력 제어를 위해 범용 화일 시스템에 기반한 캐쉬 화일 시스템이다. 또한 스트리밍 미디어를 위한 전용구조로써 전송 프로토콜의 패킷 단위 캐슁을 위한 입출력 구조를 갖는다. umcFS의 기본 구조는 화일 디스크와 umcFS의 논리 구조로 구성된다. 화일 디스크는 umcFS의 입출력의 기본 개념를 제공하고, umcFS의 논리 구조는 캐쉬 관리와 제어 및 캐쉬 저장 원칙을 제공한다.

2.2 umcFS 구조

umcFS는 화일 디스크를 기반으로 캐쉬 관리 블럭과 데이타 블럭으로 구성된다. 화일 디스크는 범용 화일 시

스템의 입출력을 제어하기 위한 가상 디스크이다. 즉 범용 화일 시스템을 디스크 드라이브 위에 논리적인 저장 구조로 구성하듯, 사용자 수준의 입출력 제어구조를 갖기 위해 디스크 드라이브에 대용량 화일을 할당함으로써 저장 공간을 선점하는 기술이다. 화일 디스크는 대용량 화일을 통한 디스크 블럭의 연속 할당과 범용 화일 시스템의 디스크 입출력 요구 큐의 관리 방식을 이용해 효율적인 입출력 관리를 지원한다. 범용 화일 시스템을 경유한 디스크 블럭의 연속 할당은 범용 화일 시스템의 제어 블럭 때문에 물리적으로 완전한 연속구조라 할 수 없지만, 범용 화일 시스템의 관점에서 논리적인 연속 할당이라 할 수 있다. 범용 화일 시스템의 디스크 입출력 요구는 물리적 디스크 드라이브 당 배정된 큐를 통해서 수행되므로 화일 디스크를 이용한 저장 공간의 선점은 디스크 드라이브에 대한 입출력 큐와 입출력 대역폭의 선점을 의미한다. 화일 디스크를 활용한 디스크의 순차 입출력은 범용 화일 시스템의 최고의 성능이라 할 수 있다.

화일 디스크의 구성에 있어 범용 화일 시스템이 지원해야할 두가지 기능이 있다. 범용 화일 시스템은 대형 화일과 대용량 화일 시스템(Large File System : LFS)을 지원해야 한다. 화일 디스크는 디스크 드라이브에 일대일 맵핑되어 저장 공간을 확보한다. 디스크 드라이브 용량은 마그네틱 고밀집도 기술의 향상으로 100GB 이상의 대용량 제품들로 출시되고 있고 계속해서 TB 용량에 도달할 것이다. 기존에 32-bit 시스템에서 범용 화일 시스템은 최고 2GB(2^{31})이상의 화일 크기를 지원하지 않기 때문에 2GB이상되는 디스크 드라이브 당 화일 디스크 할당이 불가능하다. LFS에 속하는 화일 시스템은 64-bit 시스템에 기반한 화일 시스템과 32-bit 시스템에서 64-bit 주소 공간[1]을 지원하는 화일 시스템으로 구분할 수 있다. 범용 화일 시스템의 입출력은 버퍼 캐쉬를 경유한 간접 입출력이며 스트리밍 미디어의 특성을 반영한 효율적인 입출력을 위해 Direct I/O 기능을 지원해야 한다. 버퍼 캐쉬 입출력은 블럭 저장 장치의 입출력 향상을 위해 반복해서 읽기를 수행하는 응용에 적당하고, Direct I/O는 버퍼 캐쉬를 거치지 않고 사용자 메모리 공간으로 직접 데이타를 전송하기 때문에 한번 읽어들인 데이타를 재참조할 가능성이 적은 응용에 적당하다. 예를 들어, 멀티미디어 데이타는 사용자의 순차적인 접근 방식에 따라 재참조 가능성이 적으며 대용량이기 때문에 재참조하더라도 버퍼 캐쉬에 남아 있을 가능성이 적다. Direct I/O의 버퍼 공간은 가상 메모리의 스왑 입출력을 방지하기 위해 물리적인 저장 공간에 할당되어야 한다.

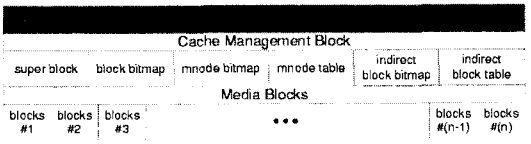


그림 2 umcFS 구조

umcFS 구조는 그림 2와 같이 umcFS의 관리를 위한 캐쉬 관리 블럭과 미디어 저장을 위한 미디어 블럭들로 구성된다. 캐쉬 관리 블럭은 super block, mnode table, indirect block과 미디어 노드를 관리하는 block bitmap, mnode bitmap, indirect block bitmap이 있다. super block은 범용 화일 시스템 정보와 umcFS의 관리를 위한 필수적인 정보를 유지한다. mnode table은 캐싱될 대상 미디어에 대한 정보를 담는 mnode의 집합이며, indirect block은 mnode에 할당할 미디어 블록을 참조하기 위한 블럭 포인터의 집합이다. 그리고 각각의 비트맵 구조는 미디어 노드와 블럭에 일대일 관계로 설정된다. super block은 운영체제의 범용 화일 시스템 정보, umcFS의 상태 정보, 나머지 제어 정보의 위치와 크기를 기술한다. 범용 화일 시스템 정보는 화일 시스템의 용량과 디스크 드라이브, 블럭 크기등이 있으며 화일 디스크를 다루는 기본적인 정보로 활용한다. umcFS의 상태 정보는 umcFS의 용량과 미디어 블럭 크기 등이 있고, 제어 정보는 super block에 이어서 오는 bitmap 구조들과 mnode table의 용량, 위치, 크기 정보를 담는다. mnode는 유닉스 화일 시스템의 inode와 유사한 구조로 미디어 식별자, 미디어 시간 정보, 캐쉬 정보, 블럭 참조 정보로 구성된다. 미디어 식별자는 umcFS 내에서 미디어에 대한 식별자로 세션 설정 과정에서 RTSP의 url 정보와 동일하고, 미디어 시간 정보는 생성과 참조에 대한 시간 정보이다. 블럭 참조 정보는 대상 mnode에 대한 할당될 블럭 포인터로 직접 포인터와 간접 포인터로 구분한다.

그림 3은 mnode 구조를 도식한 것으로 직접 블럭 참조와 확장된 간접 블럭 참조 구조를 보여주고 있다. 직접 블럭 참조는 미디어 블럭을 지정하는 일대일 맵핑 구조이며 배정된 블럭 수를 초과하는 경우 간접 블럭 참조에 블럭 참조 테이블을 배정한다. 블럭 참조 테이블은 각각 미디어 블럭에 맵핑되며 블럭 참조 테이블의 초과시에 마지막 블럭 포인터를 통해 다른 블럭 참조

1) 이론적인 최대 화일 크기는 8EB(exabyte)이다.

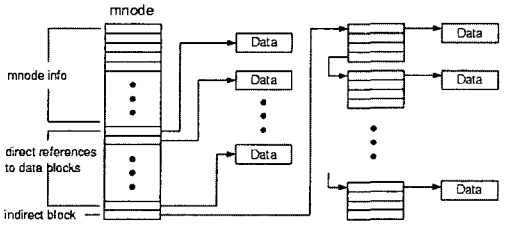


그림 3 mnode 구조

테이블을 할당받는다. 캐쉬 정보는 캐싱된 미디어의 참조를 횟수와 시간 간격으로 기록하고 캐싱 시스템의 캐쉬 관리자에 의해서 캐쉬 적중률을 반영한 등급을 유지한다.

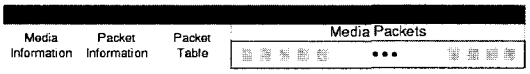


그림 4 umcFS 블록 구조

umcFS의 미디어 블럭 구조는 그림 4와 같이 헤더와 미디어 패킷들로 구성된다. 미디어 블럭의 헤더는 패킷 정보와 패킷 테이블로 구분된다. 패킷 정보는 전송 프로토콜과 블럭 내에 존재하는 전송 패킷들의 시간 범위를 기록한다. 패킷 테이블은 패킷의 블럭 내에 패킷의 위치와 각각의 패킷에 대한 시간 정보, 패킷 순서를 담는다. 이와 같은 패킷 정보에 대한 구성은 패킷의 분석 없이 헤더 정보만으로 캐쉬의 재전송을 수행할 수 있게 하며, 캐싱된 미디어에 대한 검색을 쉽게 한다.

umcFS의 미디어 블럭 관리는 블럭 자체에 국한되며, 블럭 내부 구성은 상위 캐싱 시스템에서 수행한다.

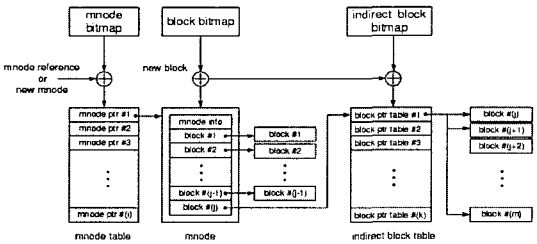


그림 5 mnode의 블럭 관리 구조

2.3 mnode 관리

그림 5는 mnode의 블럭관리 구조를 보여준다. 캐싱 시스템으로부터 새로운 미디어 등록과 저장에 대한 요구가 있을 경우, umcFS는 mnode bitmap에서 빈 mnode를 찾아 할당한다. 미디어 할당은 mnode에 포함된 직접 블럭 포인터와 block bitmap을 통해 미디어 블럭을 배정하고, 직접 블럭을 초과하는 경우에는 indirect block bitmap에서 비어있는 간접 블럭 테이블을 할당받아 미디어 블럭을 할당한다. 간접 블럭 테이블은 미디어 블럭에 대한 포인터 집합으로 범용 화일 시스템의 블럭 크기로 할당하며, 테이블의 마지막 포인터는 다른 간접 데이블의 포인터로 미디어 블럭 할당을 연속적으로 확장하기 위한 구조이다. 캐쉬 미디어의 할당은 umcFS의 범위 내에서 무제한으로 할당할 수 있다.

mnode의 직접 블럭 포인터와 간접 블럭 포인터 테이블의 할당은 umcFS를 초기화하는 과정에서 결정된다. 이와 같은 제어 블럭의 정적 배치는 유닉스 화일 시스템에서 직접 블럭을 고정적으로 할당하고 inode에 할당된 화일의 크기 변화에 따라 블럭 참조 테이블을 추가하는 것과 비교할 수 있다. 즉 umcFS는 블럭 관리 정보와 미디어 블럭을 초기에 독립적인 위치에 배정함으로써 제어 블럭을 검색하고 수정하는 부하를 줄일 수 있고, 제어 블럭이 미디어 블럭들의 중간에 배정되지 않아 미디어 블럭의 연속성을 증가시킬 수 있다.

$$MN_{vm} = B_{vm} \tag{1}$$

$$IBT_{vm} = \lceil MN_{vm} / (DBP_{vm} + 1) \rceil \tag{2}$$

$$BRP_{\text{sum}} = (DBP_{vm} + 1) \cdot MN_{vm} + (OB_{size} / BP_{size}) \cdot IBT_{\text{sum}} \tag{3}$$

$$\min(BRP_{\text{sum}}) = optimal(DBP_{vm}) \tag{4}$$

표 1 umcFS의 초기화 인자

인자	기술
OB_{size}	범용 화일 시스템의 블럭 크기
B_{size}	캐쉬 블럭의 크기
B_{vm}	umcFS의 블럭수
BP_{size}	블럭 포인터의 크기
BRP_{vm}	블럭 참조 포인터의 총 수
DBP_{vm}	mnode에 할당된 직접 참조 블록 포인터의 수
DBP_{sum}	직접 참조 블럭 포인터의 총 수
IBT_{vm}	간접 참조 테이블의 총 수
MN_{vm}	mnode의 총 수

표 1은 umcFS의 제어 블럭 할당을 위한 초기화 인자들이다. umcFS 제어 블럭은 초기화 시점에 캐쉬의 총 저장 공간에 비례한 고정 크기로 결정되고 캐쉬 입출력 중에 가감되지 않는다. 제어 블럭 크기는 mnode 테이블과 간접 블럭 참조 데이블이 대부분을 차지하고 mnode의 DBP_{vm}과 IBT_{vm}의 관계에 따라 결정된다. DBP_{vm}과

IBT_{vm}는 상충 관계에 있다. 식 1은 umcFS의 총 블럭 수에 따라 mnode 테이블 크기를 배정함을 나타낸다. 즉, mnode 당 미디어 블럭 한개를 배정한 전체 캐쉬 상태를 가정한 것이다. 식 2, 3, 4는 mnode의 직접 블럭과 간접 블럭 테이블 수의 최적화 값을 계산한다. umcFS에서 IBT_{vm}의 갯수는 식 2에서와 같이 전체 블럭 수와 mnode의 블럭 포인터 수로 계산할 수 있다. 즉, DBP_{vm} +1개의 미디어 블럭을 가진 mnode는 직접 블록에 배정을 초과한 1개의 미디어 블럭을 위해 간접 블럭 포인터를 배정해야 한다. 따라서 DBP_{vm}+1개의 미디어 블럭을 가진 mnode로 umcFS를 구성했을 때 배정할 수 있는 mnode의 최대 수로 IBT_{vm}값을 할당함으로써 umcFS의 저장 공간을 운영할 충분한 제어 블럭의 할당과 저장 공간 낭비를 방지할 수 있다. 식 3에서의 BRP_{sum}값은 mnode 테이블의 블럭 포인터 수와 간접 블록 포인터 테이블의 포인터 수를 더한 값이다. 따라서 식 1, 2와 식 3을 계산하면 BRP_{sum}와 DBP_{vm}의 함수 관계를 유도할 수 있으며 식 4의 기준에 따라 BRP_{sum}가 최소가 될 때 최적화된 DBP_{vm}와 IBT_{vm}를 구할 수 있다. 예를 들어, 범용 화일 시스템의 용량이 2TB이고, OB_{size}는 4KB 일 때, B_{size}는 512KB 이면 B_{vm}과 MN_{vm}는 4,195,304 이다. 또한 BP_{size}의 크기는 4B이다. 이 값들을 식 2, 3, 4에 적용하면 BRP_{vm}의 최소값은 DBP_{vm}이 31일 때 268,500,224이고, IBT_{sum}은 131,104 이다. 따라서 umcFS의 제어 블럭 크기는 super block(4KB), mnode table(4,195,304KB), indirect table block(524,416KB), block bitmap과 mnode bitmap(4908KB), indirect table block bitmap (20KB)들의 총합으로 대략 4.7GB이며 범용 파일 시스템의 0.22%를 차지한다.

그림 6은 BRP_{sum}와 DBP_{vm}의 함수 관계를 그래프로 표현한 것이다. 구간 [0.. Δx]에서의 BRP_{sum}감소 현상은

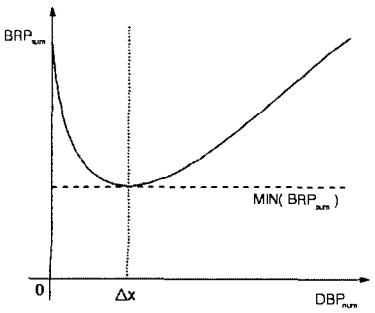


그림 6 블록 참조 포인터의 최소값

DBP_{vm}의 증가 따른 IBT_{vm}의 값이 간접 블럭 포인터 테이블 단위로 감소하기 때문이다. mnode에서 BRP_{sum}의 값이 0이면 IBT_{vm}의 값은 미디어 블럭 수와 동일하다. 구간[Δx<]에서 DBP_{vm}의 상승에 따라 대략 mnode의 정수배로 BRP_{sum}값이 증가한다.

2.4 화일 디스크 구성

umcFS는 범용 화일 시스템에 할당된 화일 디스크를 기반으로 물리적인 디스크 드라이브를 제어하는 사용자 수준의 화일 시스템이다. 화일 디스크는 디스크 드라이브와 일대일 관계를 갖기 때문에 2^{31}Bytes(2GB)이상 용량의 디스크 드라이브를 제어하기 위해서 2^{63}Bytes (8EB)의 화일 크기를 지원하는 화일 시스템이 필수적이다. umcFS의 성능은 범용 화일 시스템의 제어 블럭 구조와 화일 할당 정책과의 밀접한 관련이 있다. 범용 화일 시스템이 디스크 드라이브 위에 물리적인 위치를 관리하기 때문에 화일의 물리적인 분포는 연속성의 정도에 따라 입출력 대역폭에 영향을 미친다.

유닉스 화일 시스템의 일반적인 구조는 부트 블럭과 저장 공간을 관리하는 단위인 그룹들로 구성된다. 각각의 그룹들은 화일 시스템이 생성될 때 동일한 크기로 분할되며 독립적인 관리 체계하에 제어 블럭과 데이타 블럭으로 구성된다. 그룹의 제어 블록에는 inode와 디렉토리 테이블등으로 구성되며 저장 공간 관리의 기본적인 개념을 제공한다. inode를 통한 다단계 블럭 참조를 통해 화일에 대한 데이타 블럭의 위치를 할당하고 이론적인 화일의 최대 크기를 제한한다. 최근 화일 시스템은 일반 데이타 파일에서 테이타 베이스, 멀티미디어 등과 같은 실시간 데이타의 혼합구조를 갖거나 특수 용도의 화일 시스템으로 재구성되고 있다. 이와 같은 현상은 실시간 데이타가 갖는 시간 제한 특성을 전통적인 유닉스 화일 시스템에서는 만족시킬 수 없기 때문이다. 따라서 사용자의 요구에 따라 실시간 데이타에 대한 블럭의 연속성을 높이거나 데이타 블럭을 디스크 섹터 크기에서 수 GB까지 할당할 수 있게 한다. 또한 다단계 참조 구조의 블럭 참조 시간을 줄이고 대용량 화일을 할당하기 위해 B-tree 구조를 구성하는 경우도 있다.

그림 7은 화일 디스크 구성시 xFS[7] 저장 공간의 할당과 순서를 기술한 것으로 5.6 GB 용량의 디스크에 8 그룹들로 구성된 것으로 xfs_bmap 툴을 이용하여 얻은 값을 도식한 것이다[2]. 여기서 그룹의 순서는 물리적인 블럭의 연속성을 의미하여 그룹 안에 검은색 블럭은 제어 블럭, 회색 블럭은 제어 블럭 혹은 빈 블록이다.

2) xFS에 대한 설명은 4.1절에 자세히 기술되었음.

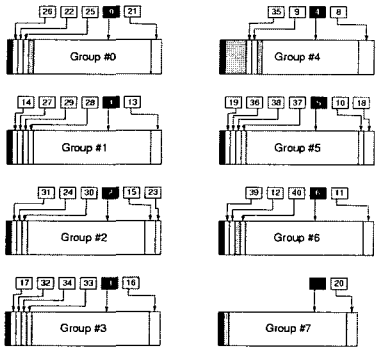


그림 7 xFS 기반의 화일 디스크 할당

그룹 위에 네모 박스로 순서를 매긴 것은 화일 디스크의 논리적 연속성과 물리적 연속성간의 맵핑 관계를 보여 주고 있다. 즉 네모 박스의 순서는 화일 디스크의 논리적인 연속성을 의미한다. 네모 박스가 검정색으로 채워 진것은 xFS가 초기에 그룹 별로 데이타 블럭을 할당할 때 최대의 연속 블럭을 얻음을 알 수 있다. 그 이후 앞, 뒤쪽에 빈 블럭들을 최종적으로 이산된 데이타 할당을 수행한다.

2.5 umcFS API

umcFS의 API는 캐쉥 시스템의 스트리밍 미디어 입출력에 대한 인터페이스다. umcFS의 API는 캐쉬 미디어의 블럭화된 입출력과 미디어 검색 그리고 캐쉬 관리 정책을 적용하기 위한 기능들로 구성되고, 전반적인 API 설계 측면에서 유닉스 화일 시스템 API와 유사하다. umcFS의 API를 세부적으로 살펴보면 다음과 같다.

• **umount ()** : 스트리밍 미디어 캐쉥 시스템은 다중 umcFS로 구조화된 화일 디스크들을 제어 리스트로 올리기 위해 umount 명령을 사용한다.

• **uopen(), uclose()** : uopen 요구는 미디어의 입출력을 위한 제어 구조를 개설하고, 새로운 미디어에 대해서 umcFS에 등록하고 자원을 할당한다. uopen 모드에는 ONLYRD, ONLYWT, APPEND로 구분된다. ONLYRD는 이미 다른 세션에서 캐쉬 콘텐츠를 사용하고 있는 경우로 캐쉬 읽기만을 제공하고 ONLYWT는 캐쉬 콘텐츠가 미동록되어 있는 경우를 의미한다. APPEND는 등록되어 있는 캐쉬 콘텐츠에 대한 쓰기와 읽기 권한을 세션에 할당한다. uclose는 캐쉥 미디어의 제어를 닫고 할당된 자원을 해제한다.

• **uread(), uwrite(), useek()** : 캐쉬의 입출력을 구성하는 기능들이다. uread는 캐쉬 입력이고, uwrite는 캐쉬 출력을 담당한다. useek의 모드는 USEEK_SET, USEEK_END, USEEK_PREV, USEEK_NEXT로 구분한다. USEEK_SET는 콘텐츠의 입출력 수행 위치를 처음으로 지정하고 USEEK_END는 끝을 지정한다. USEEK_PREV와 USEEK_NEXT는 전 블럭과 다음 블럭을 지정한다.

• **uremove(), uremove_block()** : uremove는 캐쉬 관리 정책에 따라 캐쉬 아웃이 결정될 때 호출되는 함수로 umcFS에서 미디어 리스트를 삭제한다. uremove_block은 캐쉥된 미디어에 대한 블럭 단위 삭제를 지원한다.

• **uplace(), ureplace ()** : 캐쉥 시스템의 캐쉬 관리 정책에 따라 미디어의 위치 관리와 여유 블럭이 없는 경우 블럭의 강제 아웃을 통해 캐쉬 입력을 수행한다.

• **ustatfs(), ulookup()** : statfs는 umcFS의 통계 정보를 제공하고, ulookup은 미디어와 블럭에 대한 검색 기능을 제공한다.

3. 실험 결과

다음은 본 논문에서 제시하고 구현한 umcFS의 실험 구성을 기술하고 구현 시스템의 성능을 분석한다. 실험에서는 시스템 환경과 범용 화일 시스템에 기반한 화일 블럭 기반의 스트리밍 미디어 캐쉥 시스템을 비교 모델로 제시한다. umcFS와 비교 모델과의 성능 비교는 저수준 입출력과 API 수준의 입출력을 비교 평가한다.

3.1 실험 구성

umcFS의 개발 환경은 다음과 같다. 개발 호스트는 Compaq Proliant ML370이며 Dual P-III 866 MHz에 메모리 512MB를 장착하고 있다. 사용된 운영체제는 대용량 화일 시스템(LFS)[8]을 지원하는 리눅스 시스템이다. 최근에 리눅스 시스템은 LFS의 64-bit 주소 범위을 지원하기 위해 사용자 라이브러리와 커널을 개선하고 있다. 사용자 라이브러리는 glibc 2.1.3에서 LFS을 지원하기 시작했으며, 커널은 화일 시스템에 따라 지원이 결정된다. LFS를 지원하는 화일 시스템은 JFS[9], xFS, ReiserFS[10] 등이 있다. 본 연구에서는 효율적인 저수준 입출력을 지원하기 위해 리눅스에서 Direct I/O 기능을 지원하는 xFS를 선택하였다. SGI's xFS는 멀티미디어를 위해 개발된 저널링 화일 시스템으로 대용량 화일 시스템, 입출력 대역폭 보장, 빠른 복구 등의 기능을 제공한다. 최근 xFS는 AIX 기반에서 리눅스로 포팅되었지만, 리눅스가 32-bit 운영체제이고 버퍼 캐쉬와 VFS 구조의 차이로 일부의 기능만 포팅되었다. 리눅스

에서 xFS는 성능과 용량에 대해 제한된 기능만을 제공하지만, 2^{63}Bytes(8EB) 크기의 화일과 2048 GB(2TB)의 화일 시스템 크기의 지원은 umcFS의 구성한 조건을 만족시킨다. 실험에서 사용한 스트리밍 환경은 Apple's Darwin 3.0 서버와 Quick Time Player Pro 5.0이며 실험에서 사용한 스트리밍 미디어 포맷은 Apple에서 제공하는 Hinted Track화된 MOV를 활용하였다.

실험 방법은 저수준과 API 수준에서 입출력 성능을 측정하였다. 저수준의 입출력 대역폭의 측정은 32K, 64K, 128K, 256K, 512K, 1024K, 2048K, 4096K, 8192K의 캐쉬 블럭을 할당하여 각각의 경향과 성능을 분석하고 적정 캐쉬 블럭 크기를 결정한다. 각각의 블럭별 테스트는 순차 입출력과 임의 입출력으로 구분한다. 순차 입출력은 캐싱 시스템의 전체 캐쉬 공간을 처음부터 마지막 블럭까지 연속해서 입출력을 수행하고 임의 입출력은 임의 선택된 캐쉬 블럭에 대한 일정시간 동안의 입출력 대역폭을 측정한다. API 수준에 입출력은 스트리밍 미디어 캐싱 서버에서 수용된 각각의 커넥션에 대한 입출력 대역폭과 교체시간을 측정한다. umcFS와의 성능 비교를 위해 화일 블럭에 기반한 캐싱 시스템을 구현하였고 순차 입출력과 임의 입출력을 동일하게 적용하여 실험하였다.

3.2 화일 블럭 캐쉬 비교 모델

화일 블럭 캐싱 시스템은 umcFS와의 성능 비교를 위해 개발한 캐쉬 입출력 시스템 모델이며 웹 기반의 Squid[11] 캐쉬 입출력 체계와 umcFS의 관리 체계를 도입하여 설계하고 개발하였다. 화일 블럭 캐쉬의 주요 특징은 다음과 같다. 화일 블럭의 입출력은 umcFS에서 사용한 하드웨어 장치와 운영체제 기능을 동일하게 적용하였다. 즉, DMA 장치를 활용하여 저수준의 데이타 전송을 향상시키고, 버퍼 공간을 위한 가상 메모리의 페이지 입출력 향상을 위해 가상 메모리 잠금 기능을 활용하였다. 또한 범용 화일 시스템에서 제공하는 Direct 입출력과 비동기 입출력을 도입하였다. 캐쉬 저장 공간은 범용 화일 시스템으로 초기화된 디스크 드라이브에 고정 크기의 화일 블럭을 순차적으로 생성하여 디스크 저장 공간을 선점하도록 구성한다. 화일 블럭의 순차적인 할당은 umcFS의 가상 디스크와 유사한 개념으로 범용 화일 시스템에서 화일의 저장 공간 할당 방식에 따라 화일 단위의 논리적인 연속성을 제공한다. 즉, 범용 화일 시스템이 제공할 수 있는 물리적인 연속 할당을 최대한 보장하는 것이다. 미디어 캐쉬를 위한 제어 블럭과 미디어 블럭은 같은 디스크 드라이브에 존재하지 않도록 구성하였다. 제어 블럭은 이진 화일 구조이고 화일 블럭 캐쉬를 셋업하기 위한 정보와 콘텐츠 및 화일 블럭 정보로 구성된다. 캐쉬 관리구조는 운영체제의 버퍼 캐쉬 구조를 응용하여 콘텐츠 리스트와 빈 블럭 리스트로 구성한다. 콘텐츠 리스트는 화일 블럭에 대한 리스트 구조를 내포하며 물리적인 위치에 상관없이 리스트의 순서에 따라 캐쉬 콘텐츠를 관리한다. 빈 블럭 리스트는 콘텐츠의 가감에 따라 발생하는 빈 블럭들의 집합이다.

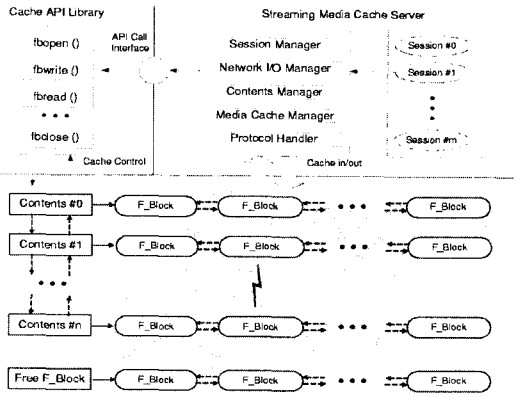
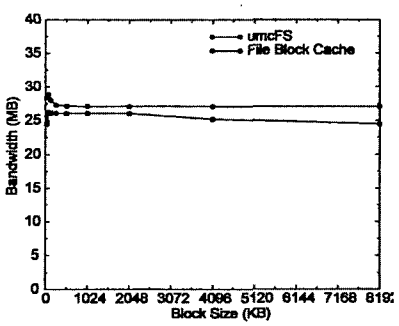


그림 8 파일 블록 캐쉬 기반의 스트리밍 미디어 캐싱 시스템

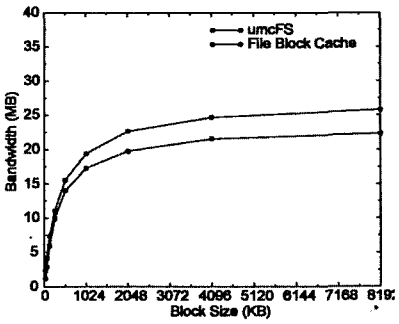
그림 8은 화일 블럭 캐쉬 기반의 스트리밍 캐싱 시스템의 구성을 보여준다. 화일 블럭 캐쉬 API와 미디어 스트리밍 캐쉬 서버는 umcFS와 동일하게 구성하였다. 화일 블럭 캐쉬는 초기에 모든 블럭을 빈 블럭 리스트에 등록하고 세션으로부터 캐싱 입출력에 대한 요구가 있으면 콘텐츠 리스트에 url과 반입된 블럭을 등록한다. 블럭 리스트는 캐쉬 입력한 블럭 순서이다. 블럭 제거는 블럭 리스트의 뒷 부분부터 수행하고 빈 블럭은 빈 블럭 리스트에서 수거한다.

3.2 저수준 입출력 성능

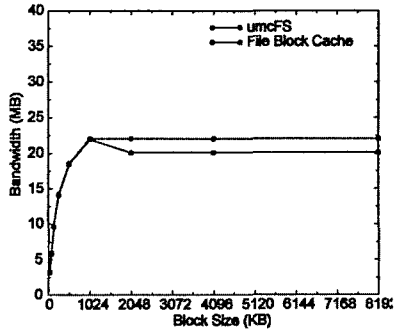
저수준 입출력 실험은 umcFS와 화일 블럭 캐쉬의 캐쉬 블럭에 대한 범용 화일 시스템에 기반한 디스크 입출력 성능을 측정한 것이다. 그림 9는 umcFS와 화일 블럭 캐쉬의 순차 입출력과 임의 입출력 대역폭을 블럭 크기 별로 측정한 결과이다. 순차 대역폭 실험은 umcFS의 첫 블럭부터 연속해서 입출력한 시간을 측정해 얻은 대역폭의 평균값이고, 임의 대역폭은 난수 발생기를 통해 지정된 블럭에 대한 입출력한 시간을 측정하여 대역폭의 평균값으로 나타내었다. 그림 9(a), 9(b)는 순차 읽



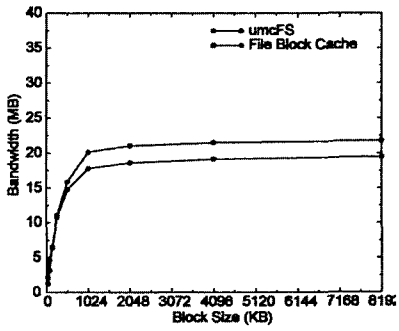
(a) 순차 읽기의 평균 대역폭



(b) 임의 읽기의 평균 대역폭



(c) 순차 쓰기의 평균 대역폭

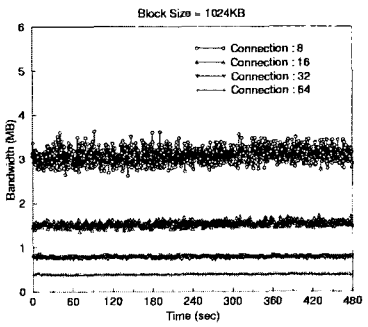


(d) 임의 쓰기의 평균 대역폭

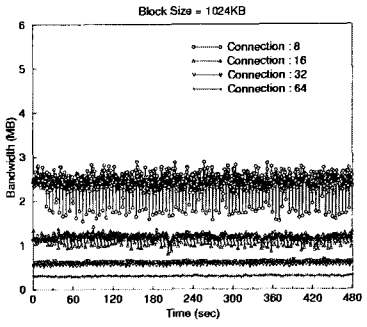
그림 9 umcFS와 화일 블럭 캐쉬의 저수준 입출력 대역폭 비교

기와 임의 읽기 대역폭을 비교한 것이다.

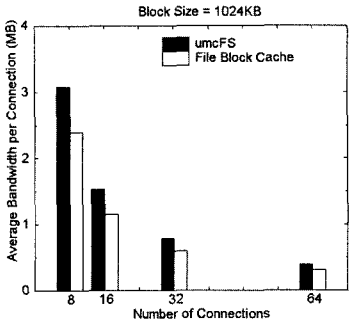
umcFS의 순차 읽기는 블럭의 크기에 따라 약간의 차이를 두고 고대역폭을 유지한다. 이것은 디스크 드라이브에서 데이타 읽기가 디스크에 장착된 디스크 버퍼를 활용한 캐쉥 정책이 적용되기 때문이다. 예측 읽기를 통해 미리 버퍼에 옮겨 놓는다면 연속 블럭의 순차 읽기 성능은 디스크의 최고 성능에 근접한 결과를 초래한다. 따라서 그림 9(a)의 결과는 디스크의 버퍼 캐쉬의 성능을 반영한 것이다. 그림 9(b)의 임의 읽기는 블럭 크기가 증가 할 수록 점차적으로 대역폭이 상승한다. 블럭 크기가 커짐으로써 블럭 단위의 연속성의 증가로 4096KB 이상의 대역폭에서 순차 읽기와 비슷한 성능을 보이고 있다. 결과는 umcFS의 미디어 블럭 크기의 결정과 입출력의 기본 정책을 제시한다. 블럭 크기가 작으면 입출력 대역폭의 효율성이 낮아지고, 블럭 크기가 커지면 캐쉬 교체 정책에 따라 캐쉬 아웃되는 미디어 블럭의 손실률이 커짐에 따라 다양한 정책의 적용과 작은 크기의 미디어에 대해 응용이 어려워진다. 그림 9(c), 9(d)는 쓰기 대역폭을 나타낸 것으로 블럭 크기의 변화에 따라 순차 읽기와 임의 쓰기의 대역폭은 유사한 경향을 보이고 있다. 그림 9(c)에서 umcFS와 화일 블럭의 쓰기 경향이 1024KB까지 근소한 차이로 상승하다가 그 이후에 차이를 보인다. 이 현상은 화일블럭 크기가 커질 수록 블럭내에 물리적인 연속성이 약화되기 때문이다. umcFS와 화일 블럭 캐쉬의 시스템 콜 호출 경향은 입출력 성능을 결정하는 중요한 요소이다. umcFS는 화일 디스크의 마운트와 언마운트 시에 open과 close를 한번 수행하고 블럭 입출력에 대해 read/write 호출을 수행한다. 반면에 화일 블럭 캐쉬는 블럭 입출력을 수행할 때마다 open, close와 read/write 콜을 수행한다. 즉, umcFS는 범용 화일 시스템의 자원에 대한 선점 구조이고, 화일 블럭 캐쉬는 자원의 설정과 해제를 범용 화일 시스템에 의존하는 비선점 구조이기 때문에 성능 차이가 발생한다. 캐쉬 입출력은 순차적으로 발생하지 않는다. 매번 발생하는 캐쉬 입출력에 따라 콘텐츠에 대한 블럭 사이에 순차성이 사라지기 때문에, 캐쉬 입출력 성



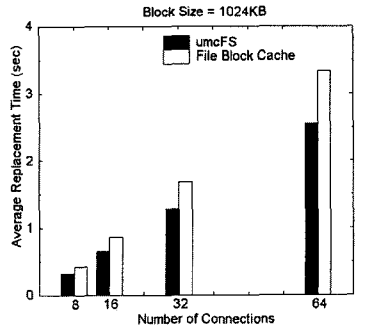
(a) umcFS의 교체 대역폭 경향



(b) 화일 블럭 캐쉬의 교체 대역폭 경향



(c) 캐쉬 평균 교체 대역폭



(d) 캐쉬 평균 교체 시간

그림 10 umcFS와 화일 블럭 캐쉬의 API 입출력 대역폭 비교

능은 임의 입출력 경향을 반영한다. umcFS의 적정 미디어 블럭 크기는, 블럭 크기별 입출력 효율성에 관한 그림 9의 실험 결과와 큰 블럭 크기에 대한 단편화 문제를 고려할 때 1024KB가 적절하다고 판단된다. 이러한 1024KB의 블럭 크기에 대한 umcFS의 임의 입출력 성능은 화일 블럭 캐쉬에 비해 13% 성능 향상을 보였다. umcFS와 화일 블럭 캐쉬의 성능 차이는 시스템 콜을 통한 입출력 제어 구조가 주요 원인이며 부가적으로 화일 블럭 캐쉬에서 캐쉬 블럭이 커짐에 따라 물리적인 연속성이 약화되는 현상도 영향을 미친다.

3.3 API 수준 입출력 성능

API 수준의 입출력은 사용자 세션의 캐쉬 입출력이며, umcFS에서는 uread와 uwrite 콜을 통해 수행된다. 실험 설정은 umcFS와 화일 블럭 캐쉬에 동일하게 적용하였다. 콘텐츠 당 10개의 블럭을 할당하여 캐쉬 전체를 채웠으며 각 스트리밍 커넥션은 다른 커넥션과 중복없이 임의로 콘텐츠를 선택한다. 콘텐츠 요구 증가에 따른 캐쉬 블럭의 교체는 커넥션이 설정되지 않은 콘텐츠에 대해서 임의 선택하여 마지막 블럭을 교체하였고 각각의 측정 시간은 480초로 제한하였다.스트리밍 미디어 캐쉬 서버의 구조는 umcFS와 화일 블럭 캐쉬에 동일하게 구성하였다. 입출력 큐 구성은 입출력 관리자와 커넥션들의 생산자와 소비자 관계로 구성하고 큐와 제어 데이타의 일관성 유지를 위해 mutex를 활용해 동기화하였다.

그림 10은 umcFS와 화일 블럭 캐쉬의 API 수준의 입출력 경향과 성능에 대한 결과를 보여준다. 캐쉬 교체 시간은 커넥션에 발생한 캐쉬 입력에 대해서 기존에 캐싱된 블럭을 캐싱 아웃시키고 새로운 블럭 입력에 드는 시간이고, 캐쉬 교체 대역폭은 캐쉬 블럭 크기에 대해 캐쉬 교체 시간으로 나눈 커넥션 당 입출력 대역폭이다. 그림 10(a), 10(b)는 1024KB 블럭에 대해서 커넥션 수에 따른 대역폭 변화를 보여주고 있다. umcFS는 화일 블럭 캐쉬에 비해 높은 대역폭을 제공하고 있으며 그 경향도 상대적으로 안정되어 있다. 화일 블럭 캐쉬의 입출력 경향은 저수준 입출력에서 화일 자원의 설정과 해제에 드는 비용이 반영되어 umcFS에 비해 편차가 크다. 그림 10(c), 10(d)는 1024KB 블럭 크기에 대해 커

넥션 수의 증가에 따른 교체 대역폭과 교체 시간을 나타내고 있다. 그림 10(c)는 커넥션 수의 증가에 대하여 각각의 커넥션에 대한 평균 대역폭이 비례적으로 감소하고, 그림 10(d)는 대기큐의 처리 지연 시간에 따라 평균 교체 시간의 상승을 보여주고 있다. umcFS는 1024KB 블럭 크기에서 화일 블럭 캐쉬에 비해 약 24% 성능 향상을 보였다. API 수준에서 성능 차이는 저수준 입출력 경향이 주요인이며, umcFS의 비트맵 구조와 화일 블럭 캐쉬의 리스트 구조도 입출력 성능 차이에 영향을 준다.

4. 결 론

본 논문에서는 스트리밍 미디어의 효율적인 캐쉬 저장과 재전송을 목적으로 범용 화일 시스템에 기반한 사용자 수준의 스트리밍 미디어 캐쉬 화일 시스템인 umcFS를 설계하고 구현하였다. umcFS의 입출력 구조는 범용 화일 시스템을 기반으로 화일 디스크를 구성함으로써 범용 화일 시스템이 제공할 수 있는 물리적인 저장 공간의 연속성을 극대화하였고, 64-bit 주소 범위와 Direct I/O, 메모리 잠금 기능을 활용하여 효율적인 입출력 체계를 구성하였다. umcFS의 캐쉬 블럭 구조는 1차 간접 블럭 참조와 간접 블럭 확장 구조로 단순화하였으며, umcFS의 설정 시 제어 블럭을 정적 할당하여 화일 시스템의 제어 부하를 최소화하였다. 실험에서 범용 화일 시스템의 제어 블럭은 전체 용량에 약 1.6%를 차지하고, 반면에 umcFS의 제어 블럭은 약 1%를 차지하였다. 그러나 범용 화일 시스템의 제어 블럭은 초기 설정치이기 때문에 화일에 대한 저장 공간이 할당된다면 증가될 것이다. umcFS는 사용자 수준의 라이브러리로 구성되어 개발 기간의 단축과 시스템 간에 이식성 및 확장성을 높였다. umcFS의 입출력 성능 실험은 화일 블럭 캐쉬를 비교 모델로하여 저수준과 API 수준에서 측정하였다. 실험 결과, 블럭 크기에 따른 입출력 효율성과 블럭 단편화 문제를 고려할 때 1024KB 크기의 블럭이 적정한것으로 판단하였고, 이경우 캐쉬의 임의 입출력 특성을 반영한 실험에서 화일 블럭 캐쉬보다 약 13%의 성능 향상을 보였다. 두 모델의 성능 차이는 입출력 자원 관리 구조에서 나타나는 시스템 콜의 빈도에 따른 가중된 부하가 주 원인이다. API 수준의 입출력 성능은 저수준의 입출력 경향과 커넥션 수 증가로 인한 입출력 대기 시간 지연에 영향을 받으며, umcFS의 블럭 교체 성능은 블럭 크기가 1024KB일 때 화일 블럭 캐쉬에 비해 약 24% 성능 향상을 보였다.

umcFS는 스트리밍 미디어의 패킷 단위 네트워크 입출력에 대한 전형화된 디스크 기반의 캐쉬 블럭 입출력 구조를 제시한다. 기존의 캐쉬 입출력 분야는 버퍼 캐쉬를 활용한 효율적인 전송 구조와 캐쉬 교체 정책에 대한 연구에 집중해 왔고, 상용화에 성공한 업체들은 특성화된 전용 장비의 개발과 비공개를 원칙으로 하고 있다. umcFS 캐쉬 입출력 시스템은 캐싱 서버의 성능 향상을 위한 핵심 부분을 이루며 캐싱 미디어의 형식과 활용 방식을 제시하고 있어 스트리밍 미디어 캐싱의 연구 분야에 파급효과가 있을 것으로 예상한다. 또한 umcFS 화일 디스크는 범용 화일 시스템을 활용한 저장 공간과 대역폭 선점에 대한 기술로 사용자 수준의 입출력 제어의 가능성을 보여주며 멀티미디어와 데이타 베이스 등의 비슷한 응용에도 쉽게 적용할 수 있을 것이다.

향후 과제는 본 연구에서 개발한 umcFS와 프로토콜 수준의 스트리밍 미디어 캐싱 및 재전송 기술을 포함한 스트리밍 미디어 캐싱 시스템의 개발에 있어 효율적인 캐쉬 교체 정책의 개발이다. 또한 umcFS의 안전성과 복구 체제를 위해 저널링 기능의 개발이 필요하다.

참 고 문 헌

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox. 'Caching Proxies: Limitations and Potentials,' *Proceedings of 1995 World Wide Web Conference*, Boston, 1995, pp.119-133.

[2] Yeuwei Wang, Zhi-Li Zhang, David H.D. Du, and Dongli Su, 'A Network Consious Approach to End-to-End Video Delivery over Wide Area Networks Using Proxy Servers,' *IEEE Infocom*, April 1998, pp.660-667.

[3] Wang, C. B., Goebel, V. and Plagemann, T., 'Techniques to Increase Disk Access Locality in the Minorca Multimedia File System,' *Proceedings of the Seventh ACM Multimedia Conference*, Orlando, October 1999, pp.147-150.

[4] W. Lee, D. S, D. Wijesekera, J. Srhastava, D. Kenchammana-Hosekotet and M. Foresti, 'Experimental Evaluation of PFS Continuous Media File System,' *Proceedings of 6th ACM International Conference of Information and Knowledge Management*, Las Vegas, November 1997, pp. 246-253.

[5] T. N. Niranjan, Tzi-cker Chiueh, Gerhard A. Schloss, 'Implementation and Evaluation of a Multimedia File System,' *IEEE International Conference on Multimedia Computing Systems*, Ontario, 1997, pp.269-276.

[6] Asit Dan and Kinkar Sitaram, 'A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads,' *Proceedings of IS&T SPIE Multimedia Computing and Networking Conference*, San Jose, January 1996, pp.344-351.
[7] *SGI's xFS File System for Linux*, URL : http://oss.sgi.com/projects/xfs.
[8] *Large File Support in Linux*, URL : http://www.suse.de/~aj/linux_lfs.html.
[9] *IBM's Journaled File System for Linux*, URL: http://www-124.ibm.com/developer/open-source/jfs.
[10] *ReiserFS : Journaled File System Based on Balanced Tree Algorithms*, URL : http://www.reiserfs.org.
[11] *Squid Web Proxy Cache*, URL : http://www.squid-cache.org.

오 재 학
정보과학회논문지 : 시스템 및 이론
제 29 권 제 5·6 호 참조

차 호 정
정보과학회논문지 : 시스템 및 이론
제 29 권 제 5·6 호 참조