버추얼 인터페이스 아키텍처 및 인터벌 캐쉬에 기반한 분산 VOD 서버

(A Distributed VOD Server Based on Virtual Interface Architecture and Interval Cache)

오 수 철 † 정 상 화 ††
(Soo-Cheol Oh) (Sang-Hwa Chung)

요 약 본 논문에서는 VIA(Virtual Interface Architecture) 통신 프로토콜과 인터벌 캐쉬 기법을 적용하여 서버 내부 통신망의 부하를 감소시킨 PC 클러스터 기반 분산 VOD 서버를 제안한다. 분산 VOD 서버의 각 노드는 클러스터상에 분산 저장된 비디오 데이타를 서버 내부 통신망을 사용하여 전송받아 사용자에게 제공한다. 이 때, 대량의 비디오 데이타가 서버 내부 통신망을 통하여 전송됨으로 서버 내부 통신망에 부하가 증가한다. 본 논문에서는 TCP/IP의 통신 오버헤드를 제거한 사용자 수준 통신 프로토콜인 VIA에 기반한 분산 VOD 파일 시스템을 개발함으로써, 원격 디스크를 접근하는데 소요되는 내부 통신망 비용을 최소화하려고 하였다. 또한, VIA의 최대 전송 크기를 VOD 시스템에 맞게 확장함으로써 내부 통신망의 성능을 향상시키려고 하였다. 추가로 본 논문은 인터벌 캐쉬 기법을 적용하여 원격 서버 노드에서 전송 받은 비디오 데이타를 지역 노드의 메인 메모리에 캐쉬함으로써, 서버 내부 통신망에 발생하는 통신량을 감소시켰다. 실험을 통하여 분산 VOD 서버의 성능을 측정하였으며, TCP/IP에 기반하고 인터벌 캐쉬를 지원하지 않는 기존의 분산 VOD 서버와 성능을 비교하였다. 실험결과, VIA 적용으로 약 11.3%의 성능 향상, 그리고 인터벌 캐쉬 기법을 적용하여 추가로 약 10%의 성능 향상이 생겨 총 21.3%의 성능 향상을 얻을 수 있었다.

키워드 : 주문형비디오, Virtual Interface Architecture, 캐쉬, 클러스터 시스템

Abstract This paper presents a PC cluster-based distributed VOD server that minimizes the load of an interconnection network by adopting the VIA communication protocol and the interval cache algorithm. Video data is distributed to the disks of the distributed VOD server and each server node receives the data through the interconnection network and sends it to clients. The load of the interconnection network increases because of the large amount of video data transferred. This paper developed a distributed VOD file system, which is based on VIA, to minimize cost using interconnection network when accessing remote disks. VIA is a user-level communication protocol removing the overhead of TCP/IP. This papers also improved the performance of the interconnection network by expanding the maximum transfer size of VIA. In addition, the interval cache reduces traffic on the interconnection network by caching, in main memory, the video data transferred from disks of remote server nodes. Experiments using the distributed VOD server of this paper showed a maximum performance improvement of 21.3% compared with a distributed VOD server without VIA and the interval cache, when used with a four-node PC cluster.

Key words : Video On Demand, Virtual Interface Architecture, Cache, Cluster System

· 이 논문은 정부(교육인적자원부)의 재원으로 한국학술진흥 재단의 지원을 받아 수행된 연구임(과제번호 D00595)
† 정 회 원 : 한국전자통신연구원 선임연구원
ponylife@etri.re.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
shchung@pusan.ac.kr
논문접수 : 2005년 8월 1일
심사완료 : 2006년 6월 16일

1. 서 론

최근에 인터넷의 급속한 보급 및 전송 속도 향상으로 인하여, 인터넷을 활용하는 다양한 형태의 멀티미디어 서비스가 급증하고 있다. 특히 가장 대표적인 멀티미디어 서비스인 VOD(Video On Demand : 주문형 비디오)는 방송국과 가상 캠퍼스 등의 여러 분야에서 수요

가 급속히 증가하고 있으며, 앞으로는 가정용 비디오 대여 시장을 대체할 정도로 시장이 커질 것으로 예상된다.

기존의 VOD 서버와 관련된 가장 기본적인 구조는 복사기반 VOD 서버를 들 수 있다. 복사기반 서버는 기존의 사용자 요구를 분석하여 서비스 요구가 많은 비디오 데이타를 복수의 디스크에 복사하여 저장하며, 각 서버 노드는 자신의 디스크에 저장된 비디오 데이타만을 사용자에게 전송할 수 있다. 따라서, 사용자의 요구가 동적으로 변하여 특정 비디오 데이타에 대한 사용자의 요구가 집중되어 비디오 데이타를 소유한 서버 노드의 용량을 초과하면, 새로운 서버 노드에 비디오 데이타를 복제한 후에 사용자의 요구를 서비스할 수 있다. 복사기반 VOD 서버에서는 이러한 과정이 offline 상에서 발생하기 때문에 동적으로 변하는 사용자의 요구를 실시간으로 수용할 수 없다. 또한, 동일한 비디오 데이타를 복수의 디크스에 복사하므로 디스크 자원의 사용 효율이 낮다는 단점이 있다.

복사기반 VOD 서버의 단점을 해결한 것으로 분산 VOD 서버가 있으며, University of Southern California의 Yima[1], ETRI의 CrownFS[2], European RDF project의 VoDKA[3]를 예로 들 수 있다. 이 방식은 비디오 데이타를 VOD 서버를 구성하는 각 노드의 디스크에 분산 저장한다. 그리고, 서비스 요청이 있을 때마다 해당 데이타가 저장되어 있는 노드의 디스크에서 서버 내부 통신망을 통하여 비디오 데이타를 전송받아서 사용자에게 제공한다. 즉, 분산 VOD 서버의 각 서버 노드는 자신이 보유하고 있지 않은 비디오 데이타도 서버 내부 통신망을 사용하여 전송받아서 사용자에게 제공할 수 있기 때문에, 복사 기반 VOD 서버와 달리 특정 비디오에 집중되거나 동적으로 변하는 사용자의 요구를 실시간으로 처리할 수 있다. 분산 VOD 서버의 단점은 서버 내부의 통신망을 통하여 대량의 비디오 데이타를 전송하므로 서버 내부 통신망에 부하가 증가하여 전체 시스템의 성능이 저하될 가능성이 높다는 것이다. 따라서, 분산 VOD 서버의 성능을 향상시키기 위해서는 서버 내부 통신망에 발생하는 부하를 최소화하는 것이 중요하다.

본 논문에서는 분산 VOD 서버가 동적으로 변화하는 사용자들의 서비스 요구를 수용할 수 있고 자원 활용 효율이 높기 때문에 VOD 서버를 위한 기본 구조로 채택하였다. 본 논문에서는 분산 VOD 서버를 구성하기 위해서 가격대 성능비가 높은 Gigabit Ethernet을 내부 통신망으로 사용한 PC 클러스터 시스템을 활용하였으며, 분산 VOD 서버의 내부 통신망에서 발생하는 부하를 감소시키기 위해서 VIA(Virtual Interface Architecture)[4]와 인터벌 캐쉬 기법[5]을 적용하였다. Gigabit Ethernet을 위한 대표적인 통신 프로토콜인 TCP/IP는 통신에 소요되는 소프트웨어 오버헤드가 많아서 네트워크의 성능이 저하되는 단점이 있으며, 이를 해결하기 위해서 등장한 것이 사용자 수준 통신 프로토콜의 표준인 VIA이다. 본 논문에서는 VIA에 기반한 분산 VOD 파일 시스템을 개발함으로써, 원격 디스크를 접근하는데 소요되는 내부 통신망 비용을 최소화 하였으며, 분산 VOD 서버의 각 노드에 존재하는 디스크들을 하나의 논리적 디스크로 관리할 수 있게 하였다. 또한, VIA에서 한번의 send/receive 명령으로 전송 가능한 최대 데이타 크기를 VOD 서비스에 맞게 확장함으로써 내부 통신망의 성능을 향상시켰다. VOD 서비스의 경우 신작 및 인기 비디오에 대한 서비스 요구가 집중될 것으로 예상되며, 이 경우 동일한 비디오 데이타가 네트워크 통하여 중복적으로 전송되기 때문에 네트워크에 부하가 많이 발생하게 된다. 이때 인터벌 캐쉬를 적용하여 지역 및 원격 디스크에서 전송받은 비디오 데이타를 효율적으로 캐쉬함으로써 통신망에 중복적으로 발생하는 통신량을 감소시켰다. 따라서, 본 논문에서는 VIA와 인터벌 캐쉬 기법을 적용함으로써 분산 VOD 서버에서 동시에 서비스 가능한 동시 비디오 스트림의 수를 대폭 향상시켰다.

2. 관련연구

기존의 분산 VOD 서버와 관련된 연구로는 Yima[1], CrownFS[2] 그리고 VoDKA[3]가 있다. 이들 시스템은 PC 클러스터 시스템에 기반하고 있으며, Yima와 VoDKA의 각 노드는 Fast Ethernet을 사용하여 연결되어 있고, CrownFS의 각 노드는 클러스터 시스템을 위한 고속 네트워크인 Myrinet[6]을 사용하여 연결되어 있다. Yima와 CrownFS의 각 노드는 디스크를 관리하면서 동시에 비디오 데이타를 사용자에게 전송하는 역할을 담당한다. 분산 저장된 비디오 데이타는 서버 내부 통신망을 사용하여 각 서버 노드에 전송되고, 각 서버 노드는 이를 하나의 스트림으로 병합하여 사용자에게 전송한다. VoDKA의 서비스 방식도 Yima나 CrownFS와 유사하지만, 가장 큰 차이점은 데이타를 저장하는 노드와 사용자에게 비디오 데이타를 전송하는 노드가 분리되어 있다는 것이다. 이러한 VOD 서버들은 디스크로부터 읽어온 데이타를 임시적으로 저장하는 버퍼는 지원하지만, 메인 메모리에 캐쉬하지는 않는다.

위에서 설명한 연구와 함께 분산 VOD 서버의 성능 향상을 위한 다양한 연구도 진행되었다. VOD 서버는 I/O 중심의 작업임으로 VOD 서비스를 수행하기 위해 필요한 주요 I/O 자원인 디스크, 서버 내부 통신망 및 사용자 통신망 등의 자원을 효율적으로 사용하여 성능

을 향상시키는 것이 중요하다. 따라서 이와 관련된 연구들에 대해서 살펴보도록 하겠다.

디스크의 성능 향상을 위한 연구로 스트라이핑 기법이 있다. 스트라이핑은 비디오 데이타를 디스크에 저장하기 위한 가장 일반적인 정책이며, 이의 성능 향상을 위한 것으로 가중치 스트라이핑 기법[7]이 있다. VOD 서버가 보유한 각 디스크들은 확장 및 유지 보수등의 이유로 각 디스크의 성능은 차이가 날 수 있다. 가중치 스트라이핑 기법은 성능이 좋은 디스크에는 가중치를 높게 부여하여 더 많은 비디오 데이타 블록을 저장하는 방식으로, 각 디스크의 성능을 고려한 부하균등화를 수행할 수 있기 때문에 디스크의 성능이 향상된다. 또한, 개선된 가중치 스트라이핑 기법[8]은 비디오에 반영된 인기도를 디스크의 가중치를 계산하는데 추가하여 가중치 계산을 좀 더 정확히 함으로써 스트라이핑 기법의 성능을 개선시키려고 하였다.

VOD 서버에서 사용자에게 비디오 데이타를 전송하는 사용자 통신망의 부하를 줄여주는 가장 기본적인 방법으로 broadcast/multicast[9,10]가 있다. 그러나, broadcast/multicast는 지연시간(비디오를 요청한 시간과 실제로 비디오 데이타가 전송되어 재생되는 시간차이)이 커서 VOD 서비스에 적절하지 못하다. 사용자 통신망의 부하를 줄이는 다른 기법으로 proxy 서버[11]와 cooperative cache[12]가 있다. Proxy 서버는 VOD 서버와 사용자를 연결하는 통신망 상에 존재한다. Proxy 서버는 VOD 서버에서 사용자에게 전송되는 비디오 데이타를 저장하고 있으며, 동일 비디오에 대한 사용자의 요청이 있을 경우, 이를 VOD 서버와의 연결 없이 바로 proxy 서버에서 사용자에게 비디오 데이타를 전송한다. 이 경우, 동일 비디오 데이타에 대한 처리를 proxy 서버가 대신해 줌으로써 사용자 통신망과 VOD 서버의 부하를 상당히 감소시켜 준다. 이와 유사한 방법으로 proxy 서버 대신 사용자에 저장된 비디오 데이타를 활용하는 cooperative cache도 있다. 비디오 데이타를 전송 받은 사용자는 비디오 데이타를 캐쉬하고 있으며, 다른 사용자가 동일한 비디오 데이타를 요구할 때 자신이 캐쉬하고 있는 비디오 데이타를 직접 전송하게 된다. 즉, proxy 서버방식과 마찬가지로 사용자 통신망과 VOD 서버의 부하를 많이 감소시켜준다.

서버 내부 통신망의 성능 향상에 관련된 연구로는 캐쉬 기법[13,14,5]과 부하 균등화 기법[15]이 있다. 비디오 데이타는 크기가 크고, 순차적으로 접근되는 특징을 가지고 있으므로 일반적인 데이타 캐쉬 기법을 적용하기 어렵다. 따라서 viewer enrollment window[13], basic[14] 및 인터벌 캐쉬[5]와 같이 비디오 데이타를 위한 전용 캐쉬 알고리즘들이 개발되었다. Viewer enrollment window는 동일 비디오 데이타에 대한 스트림이 여러 개 있을 때, 첫번재 스트림에 버퍼를 크게 할당하여 이후의 스트림들이 디스크가 아닌 버퍼에서 서비스 될 수 있도록 한 것이다. 이 방식은 단순히 버퍼를 크게 설정하는 방식을 적용함으로써 메모리가 효율적으로 사용되지 못하는 단점이 있다. Basic은 비디오 데이타가 디스크에서 읽혀질 때, 데이타와 데이타의 접근 시간을 함께 캐쉬에 저장한다. 그리고, 캐쉬 replacement를 수행할 때 접근 시간이 가장 오래된 데이타를 victim으로 선택하여 캐쉬 replacement를 수행한다. 이 방식은 사용자에 의해 접근 되는 새로운 비디오 데이타를 무조건 캐쉬함으로, 캐쉬 알고리즘이 비디오의 인기도를 반영하지 못하는 단점을 가지고 있다. 인터벌 캐쉬는 같은 비디오에 대한 스트림 요구가 연속적으로 발생할 때, 연속한 두 스트림 요구의 시간 간격에 해당하는 비디오 데이타를 캐쉬하는 정책이다. 이 알고리즘은 성능이 좋기 때문에 널리 사용되고 있다. 부하 균등화 기법[15]은 분산 VOD 서버에서 부하 균등화 작업을 수행할 때, 서버 내부 통신망의 비용이 가장 작은 서버 노드를 선택하도록 함으로써 서버 내부 통신망의 성능을 향상시키려는 연구이다.

인터벌 캐쉬는 제안된 이후로 꾸준히 추가 연구가 진행되어 왔다[16-18]. [16]은 인터벌 캐쉬에 LFU(Least-frequency-used) 알고리즘을 조합하여 인터벌 캐쉬의 성능을 향상시키려고 하였다. 인터벌 캐쉬의 단점은 새 스트림 요구시 비디오 데이타의 앞부분이 캐쉬되지 않는다는 것이다. [17]은 기존의 스트림 요구 기록에 근거한 virtual interval을 생성함으로써 비디오 데이타의 앞부분이 추가로 캐쉬되도록 하여 성능을 향상시키려고 하였다. 또한, [18]은 VOD 시스템의 디스크에 존재하는 버퍼 캐쉬를 관리하는 알고리즘으로 인터벌 캐쉬를 적용함으로써 VOD 서버의 성능을 향상시키는 연구를 진행하였다.

VIA의 경우, 현재까지 VOD 서버에 적용된 사례는 찾아보기 힘들지만, 다른 분야에서 지속적으로 사용되어 왔다. [19]는 클러스터 시스템에서 NBD(network block device)를 구축하기 위한 통신 라이브러리로 VIA를 활용하였다. 또한, [20]은 클러스터 시스템을 위한 대표적인 공유 메모리 시스템인 Treadmark의 통신 라이브러리로 VIA를 적용하여 시스템의 성능을 향상시켰다. VIA의 API는 대표적인 통신 *interface*인 *socket* interface와 다르다는 단점을 가지고 있다. [21]과 [22]는 이를 해결하기 위해서 VIA가 socket interface를 지원하도록 개발함으로써, 기존의 socket interface 기반 프로그램들이 프로그램을 재작성하지 않고도 VIA의 성능 향상을 활용할 수 있게 하였다.

3. 분산 VOD 서버

3.1 분산 VOD 서버의 구조

본 논문에서 제안하는 분산 VOD 서버의 구조는 그림 1과 같다. 분산 VOD 서버는 PC 클러스터 시스템을 기반으로 하였으며, 각 노드를 연결하는 내부 통신망으로 가격대 성능비가 높은 Gigabit Ethernet을 사용하였다. 각 노드는 Pentium 기반 시스템을 사용하였으며, 메인 메모리, SCSI 디스크, 서버 내부 통신망용 Gigabit Ethernt 카드, 사용자 통신망용 Gigabit Ethernet 카드로 구성된다. SCSI 디스크는 서비스할 비디오 데이타를 분산 저장하며, 각 서버 노드는 서버 내부 통신망용 Gigabit Ethernet을 사용하여 VOD 서버의 각 SCSI 디스크에 분산 저장된 비디오 데이타를 읽어온다. 읽어들인 비디오 데이타는 메인 메모리에 존재하는 비디오 캐쉬에 캐쉬되며, 사용자용 Gigabit Ethernet을 사용하여 사용자에게 전송된다. 본 논문에서는 서버 내부 통신망에서 발생하는 오버헤드를 감소시키기 위해서 2가지 기법을 적용하였다. 첫째, TCP/IP의 오버헤드를 제거한 사용자 수준 통신 프로토콜인 VIA를 적용하여 통신망을 통하여 비디오 데이타를 전송하는 시간을 최소화하였다. 둘째, 인터벌 캐쉬를 적용하여 지역 및 원격 디스크에서 전송받은 비디오 데이타를 캐쉬함으로써 통신망에 중복적으로 발생하는 통신량을 자체를 감소시켰다. 이러한 2가지 기법을 적용하여 분산 VOD 서버의 내부 통신망을 통하여 비디오 데이타를 전송하는 시간을 최소화함으로써, 동시에 서비스 가능한 비디오 스트림 수를 증가시켰다.

사용자가 분산 VOD 서버에 접속하여 원하는 비디오 데이타를 전송받는 과정을 설명하면 다음과 같다. 사용자는 분산 VOD 서버의 제어 노드에 접속하여 새로운 비디오의 전송을 요청한다(그림 2의 ①). 분산 VOD 서버에 존재하는 서버 노드중의 하나는 제어 노드의 역할을 함께 수행하며, 제어 노드는 분산 VOD 서버의 관리 및 사용자의 비디오 전송 요구를 처리한다. 또한, 제어 노드는 요청된 비디오가 현재 다른 사용자에게 서비스되고 있는지, 새로운 스트림 서비스를 제공할 수 있는지 등의 서버 전체 상태를 고려하여 서비스 가능한 서버 노드를 선택하고, 선택한 서버 노드로 새로운 비디오의 전송 요구 정보와 사용자 정보를 전달한다(그림 2의 ②). 비디오 전송 요구를 받은 서버 노드는 새로운 비디오 스트림 서비스를 위해서 메인 메모리 및 디스크 자원을 할당하고, 사용자와 연결을 설정한다(그림 2의 ③). 이러한 비디오 스트림 초기화 과정이 끝나면 사용자는 제어 노드가 아닌 새로 연결된 서버 노드로부터 비디오 데이타를 전송 받는다(그림 2의 ④). 서버 노드는 사용자에게 비디오 데이타를 전송할 때, 비디오 캐쉬에 해당 데이타가 존재하면 디스크 접근 없이 비디오 캐쉬에서 바로 데이타를 읽어서 사용자에게 전송한다. 해당되는

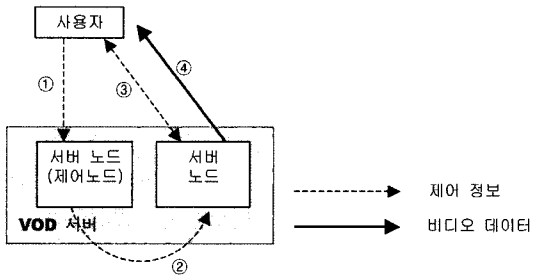

그림 2 비디오 스트림 서비스 과정

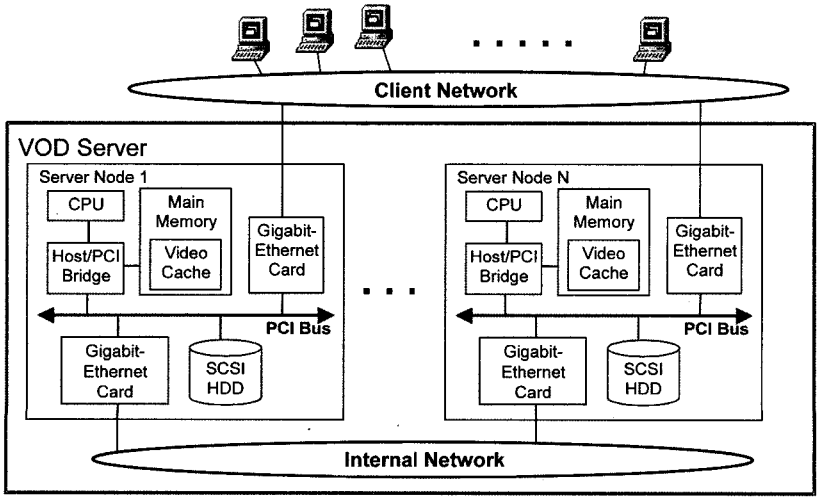

그림 1 분산 VOD 서버의 구조

비디오 데이타가 비디오 캐쉬에 없다면, 지역 및 원격 디스크에서 비디오 데이타를 읽어서 사용자에게 전송한다. 이때 각 서버 노드는 sever-push 모델[23]을 사용하여 각 스트림의 전송률이 일정하게 되도록 제어한다. Server-push 모델에서는 서버와 사용자의 연결이 설정되면, 서버는 사용자의 중지 요구가 올 때까지 정해진 전송률로 비디오 데이타를 전송한다.

3.2 VIA

본 논문에서 제안하는 분산 VOD 서버에서는 각 서버 노드들간의 비디오 데이타 및 제어 메시지를 VIA를 사용하여 전송한다. Gigabit Ethernet을 위한 가장 대표적인 통신 프로토콜인 TCP/IP는 통신시 발생하는 많은 오버헤드 때문에 통신망의 물리적 성능을 최대한으로 발휘할 수 없다. 이러한 오버헤드는 첫째, 데이타가 사용자 프로그램에서 커널 및 네트워크 어댑터로 여러 번 복사되면서 발생하며, 둘째, 데이타를 전송할 때 사용자 프로그램에서 커널로 문맥이 전환(context switch) 될 때 발생한다. 이러한 문제점을 해결하기 위해서 등장한 것이 Compaq, Intel, Microsft에서 제안한 사용자 수준 통신 프로토콜의 표준인 VIA다. VIA는 통신을 커널이 아닌 사용자 수준에서 처리하게 하여 커널로의 문맥 전환 및 커널 내부에서 소요되는 시간을 제거하였으며, 통신 계층을 단순화시킴으로써 통신 중에 발생하는 데이타 복사 회수를 최소화시켰다. 이를 통하여 VIA는 통신망의 물리적 대역폭을 최대로 활용할 수 있게 하였다.

VIA의 기본 구조는 그림 3과 같으며, 응용프로그램(application)은 VIA를 위한 API인 VIPL을 사용하여 통신을 수행한다. 응용프로그램은 VIPL을 사용하여 통신의 종단점(end-point) 역할을 하는 VI(Virtual Interface)를 생성하고, 두 노드간의 VI 연결을 위한 기능을 제공하는 VI 커널 에이전트를 사용하여 원격노드에 생성된 VI와 연결 설정작업을 한다. VIA에서는 VI 커널 에이전트를 사용하여 연결을 설정할 때만 커널이 개입되며 이후의 모든 과정은 사용자 프로그램 수준에서 처리된다. 연결이 설정된 후 VIPL을 통하여 데이타 전송 명령이 내려오면, VI 디스트립터(descriptor)를 WQ에 써 넣는다. VI 디스크립터는 네트워크 어댑터에서 처리해야 할 전송에 필요한 제어 정보 및 사용자 데이타의 주소 정보 등을 담고 있다. WQ(Work Queue)는 Send Queue와 Receive Queue로 구성되며, 데이타 송수신에 필요한 VI 디스트립터를 저장하는 역할을 하다. 이후 디바이스 드라이버는 WQ에 저장된 디스트립터를 바탕으로 VIA 통신을 위한 헤더를 생성하고, 데이타를 네트워크 카드의 MTU(Maximum Transfer Unit) 크기의 패킷으로 분할한다. 현재 Ethernet 기반 네트워트 어댑터의 MTU는 1514byte이다. 이후 디바이스 드라이버는 분할된 패킷 정보를 네트워크 업댑터의 doorbell로 전송한다. Doorbell은 I/O버스에 장착된 장치(예:네트워크 어댑터)와 CPU간에 명령과 처리 결과를 전송하는 메커니즘으로 네트워크 카드에 메모리 형태로 구현되어 있다. 즉 CPU가 doorbell에 명령어를 전송하면 네트워크 카드가 이를 인식하여 해당하는 명령어를 처리한다. 네트워크 어댑터는 doorbell에 저장된 정보에 따라서 메인 메모리에 저장된 데이타를 커널의 간섭없이 DMA를 사용하여 읽어오며, 통신망을 사용하여 원격 노드로 전송한다. 데이타를 수신한 노드는 수신된 데이타를 메인 메모리의 사용자 영역으로 DMA를 사용하여 전송하며, CQ(Completion Queue)에 데이타가 수신되었음을 표시하고, 인터럽트를 사용하여 CPU에 데이타가 수신되었음을 알려준다. CQ는 데이타를 수신 완료한 VI에 관한 정보를 저장하고 있다. 이후, CPU는 CQ에 저장된 정보를 참조하여 데이타 수신이 완료되었음을 응용프로그램에게 알려준다.

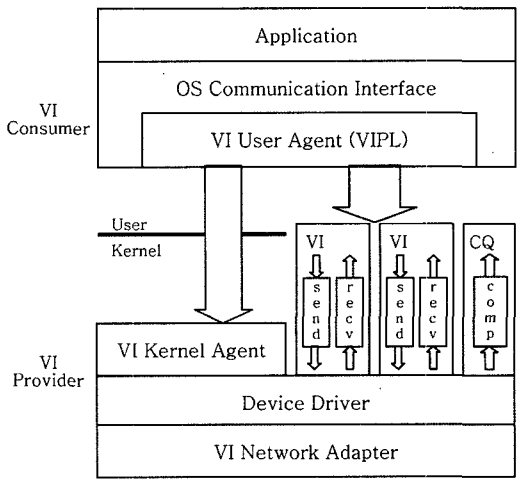

그림 3 VIA의 구조

본 논문에서는 VIA의 대표적인 소프트웨어 구현인 M-VIA[24]를 채택하였으며, M-VIA의 최대 전송 크기를 VOD 서비스에 맞게 확장하였다. M-VIA는 Ethernet 기반 네트워트 어댑터를 위해서 두 계층의 드바이스 드라이버를 지원한다. 두 계층의 디바이스 드라이버 중 상위 계층인 via_ering은 Ethernet 장치에 공통적으로 적용되는 M-VIA 기능을 처리하며, 하위 계층은 특정 네트워크 어댑터를 위한 기능을 구현한다. 본 논문의 실험에서는 네트워크 어댑터로 Intel 사의 1000/Pro Gigabit Ethernet adapter를 채택하였다. 또한, 본 네트워크 어댑터를 위한 M-VIA 용 하위 계층 디바이스 드라이버는 현재 개발되어 있지 않으므로, 본 논문에서 직

접 개발하였다. via_ering 모듈에서는 한번의 send 및 receive 명령으로 전송 가능한 데이타 크기가 32Kbyte로 제한되어 있다. 그러나, 대량의 데이타를 전송하는 VOD 서버에서 32Kbyte의 크기는 VOD 서버가 최대의 성능을 발휘하기에는 작은 크기이다. 따라서, 본 논문에서는 via_ering을 수정하여 한번의 send 명령으로 전송 가능한 데이타의 크기를 최대 1Mbyte로 확장하였으며, 이로 인하여 VOD 서버의 성능이 향상되었음을 4장의 실험에서 보일 것이다.

3.3 분산 VOD 파일 시스템

본 논문에서는 분산 VOD 서버를 위한 분산 VOD 파일 시스템을 Linux 파일 시스템과 M-VIA를 사용하여 그림 4와 같이 개발하였다. 분산 VOD 파일 시스템은 서버의 각 노드에 존재하는 디스크들을 하나의 논리적 디스크로 관리하기 때문에, 응용프로그램은 비디오 데이타가 저장된 물리적 위치에 관한 정보가 없이도 비디오 데이타를 사용할 수 있다. 분산 VOD 파일 시스템은 비디오 파일을 같은 크기의 블록으로 나누고, 각 블록을 서버 노드의 각 디스크에 스트라이핑 방식으로 저장하였다. 분산 저장된 비디오 데이타에 관한 정보는 표 1과 같이 메타 데이타에 저장된다. 메타 데이타는 비디오 데이타를 분산 VOD 서버의 각 디스크에 분산 저장할 때 생성되며, 비디오 데이타를 read할 때 각 데이타 블록의 물리적 위치를 계산할 때 사용된다. 분산 VOD 파일 시스템은 블록 단위의 파일 입출력을 지원하며, 제공하는 API는 표 2와 같다.

분산 VOD 서버는 분산 VOD 파일 시스템의 API를 사용하여 비디오 파일에 대한 read를 시도하며, 이때 read하고자 하는 데이타의 비디오 ID(movieID)와 블록 ID(blockID)를 인수로 넘겨준다. 비디오 ID는 각 비디오 파일에 대한 ID이며, 블록 ID는 비디오 파일의 초기 블록을 0으로 하여 순차적으로 증가되는 값이다. 분산 VOD 서버가 발생시킨 read 요청은 분산 VOD 파일 처리기 내부의 블록 위치 판별기로 전달된다. 블록 위치 판별기는 인수로 전달된 비디오 ID, 블록 ID 및 메타

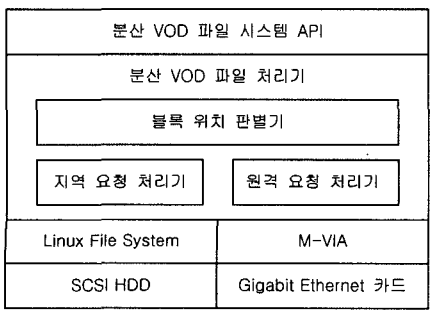

그림 4 분산 VOD 파일 시스템의 구조

표 1 분산 VOD 파일 시스템의 메타 데이타 정보

Field	설명
movieID	스트라이핑 된 비디오 파일의 ID
blkNumber	전체 블록 개수
fileSize	파일의 전체 크기
blkSize	스트라이핑 블록 크기
srtnodeID	비디오 데이타의 첫 블록을 저장하는 노드 ID
lastBlkSize	마지막 블록의 크기
N	분산 VOD 서버의 노드 수
subfiles	각 서버에 스트라이핑 된 파일의 정보 - subfileID :각 서버에 저장되는 파일의 ID - nodeID: 저장된 서버 ID - blkNumber : 서버에 저장된 블록 개수

표 2 분산 VOD 파일 시스템 API

함수	설명
dfv_open	지정된 비디오 데이타 ID에 대해 분산 VOD 서버 노드에 스트라이핑 기법으로 저장되어 있는 모든 파일을 연다.
dvf_write	스트라이핑 기법을 이용하여 분산 VOD 서버의 각 디스크에에 비디오 파일을 분산 저장한다. 지정된 데이타 블록을 지역 노드에 저장해야 할 경우 Linux 파일 시스템의 write 함수를 사용하여 지역 디스크에 저장하고, 원격 노드에 저장해야 할 경우 M-VIA를 이용하여 원격 노드로 전송하여 저장한다.
dvf_read	지정된 비디오 데이타 블록이 지역 노드의 디스크에 존재할 경우, 지역 디스크에서 바로 read 한다. 지정된 블록이 원격 노드의 디스크에 존재하면, M-VIA를 사용하여 원격 노드로부터 전송받는다.
dvf_close	지정된 비디오 파일에 대해 분산 VOD 서버 노드에 스트라이핑 기법으로 저장되어 있는 모든 파일을 닫는다.

데이타를 참조하여 실제 데이타가 분산 VOD 서버의 어떤 노드(nodeID)에 저장되어 있는지 아래식을 사용하여 판별한다.

$$nodeID = (blockID + srtnodeID) \ modular \ N$$
$$(nodeID, srtnodeID : 0 \ to \ N-1)$$

Read하고자 하는 데이타 블록이 지역 노드에 존재한다면 지역 요청 처리기로 read 요청이 전달되고, 지역 요청 처리기는 Linux file system을 사용하여 해당 비디오 데이타를 읽어온다. Read하고자 하는 데이타 블록이 원격 노드라고 판별되면, 블록 위치 판별기는 read 요청을 원격 요청 처리기로 전달하고, 원격 요청 처리기는 M-VIA를 사용하여 해당 노드로 데이타 read 요청을 전송한다. 원격 노드에 도착한 데이타 read 요청은 원격 요청 처리기로 전달되고 원격 요청 처리기는 Linux file system을 사용하여 해당 비디오 데이타를 읽어서 원격 요청을 전송한 노드로 응답을 보낸다.

3.3 인터벌 캐쉬

본 논문에서는 원격 디스크에서 읽은 데이타를 지역 노드의 비디오 캐쉬에 저장함으로써, 서버 내부 통신망에 발생하는 통신량을 감소시키려고 하였다. 비디오 데이타는 크기가 크고 순차적으로 접근되는 특징을 가지고 있기 때문에, 일반적인 데이타를 위한 캐쉬 알고리즘을 적용하기 어렵다. 본 논문에서는 비디오 데이타를 캐쉬하는 알고리즘중에서 성능이 우수한 인터벌 캐쉬[5]를 채택하였다. 인터벌 캐쉬는 같은 비디오에 대한 스트림 요구가 연속적으로 발생할 때, 연속한 두 스트림 요구의 시간 간격에 해당하는 비디오 데이타를 캐쉬하는 정책이다. 이 정책에서는 스트림의 요구 간격이 짧을수록 자주 요구되는 데이타로 간주하여 이를 캐쉬함으로써 캐쉬의 hit ratio를 높일 수 있다. 또한, 스트림의 요구 간격이 짧은 데이타는 비디오 캐쉬 요구량도 작기 때문에, 좀 더 많은 사용자 요구에 해당하는 비디오 데이타를 캐쉬할 수 있어서 캐쉬의 hit ratio는 더욱 높아진다.

본 논문에서 제안하는 분산 VOD 서버에서는 메인 메모리를 비디오 데이타를 캐쉬하는 비디오 캐쉬로 사용하였다. 또한, 비디오 캐쉬를 관리하기 위해서 interval_list, cache_list, free_list를 유지하며, 비디오 캐쉬는 디스크 스트라이핑 블록의 크기와 동일한 크기 단위로 유지된다. 인터벌 캐쉬는 동일 비디오에 대한 사용자의 스트림 요구가 연속적으로 발생할 때, 연속한 두개의 스트림 요구를 인터벌이라 정의한다. Interval_list는 현재 캐쉬되고 있지 않는 인터벌들을 캐쉬 요구량의 크기 순으로 정렬하고 있다. Cache_list는 현재 캐쉬되고 있는 인터벌들을 포함하고 있으며, free_list는 비디오 캐쉬에서 사용중이지 않는 블록들을 관리하는데 사용된다.

그림 5와 같이 동일 비디오 M_i에 대해서 스트림 요구 S_{i1}, S_{i2}, 그리고 S_{i3}가 연속적으로 발생한다고 가정하자. Si1이 서비스되고 있을 때 S_{i2}의 요구가 발생하면, 연속된 스트림 요구인 S_{i1}과 S_{i2}가 하나의 인터벌인 I_{i1}을 생성하며, 비슷한 방법으로 S_{i2}와 S_{i3}도 인터벌 I_{i2}를 생성한다. 이때, S_{i1}을 서비스하기 위해서 디스크에서 읽은 비디오 데이타는 S_{i2}를 위해서 비디오 캐쉬에 저장되며, S_{i2}는 디스크가 아닌 비디오 캐쉬에서 비디오 데이타를 읽어온다. 또한, S_{i2}도 S_{i3}을 위해서 자신이 사용한 비디오 데이타를 계속 비디오 캐쉬에 유지한다. S_{i1}이 S_{i2}를 위해서 비디오 데이타를 캐쉬할 때 필요한 비디오 데이타의 크기는 S_{i1}과 S_{i2}의 인터벌인 I_{i1}의 시간간격에 해당하는 비디오 블록 개수이다. 예를 들어 I_{i1}에 해당하는 시간간격이 3초이고 3초동안 비디오를 재생하는데 필요한 비디오 블록수가 6개라면, 6개의 블록을 캐쉬한다. 이 때, 인터벌에 해당하는 비디오 블록은 circular queue 형태로 관리되며, S_{i1}은 circular queue에 비디오 데이타를 쓰고, S_{i2}는 circular queue에서 비디오 데이타를 읽어가는 형태가 된다.

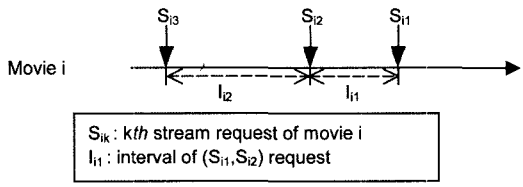

그림 5 인터벌 캐쉬의 예

새로운 인터벌이 생성되었을 때, 그 인터벌에 해당하는 비디오 데이타가 캐쉬될지의 여부를 결정하는 과정은 그림 6에 나와 있다. 비디오 M_i에 대한 (m+1)번째 스트림 요구가 발생할 때, 새로운 인터벌 I_{im}이 생성되어 interval_list에 추가된다. I_{im}이 interval_list에서 크기가 가장 작은 인터벌이 아니라면, I_{im}은 비디오 캐쉬에 캐쉬되지 않는다. Interval_list에 있는 인터벌들은 각 인터벌을 캐쉬하기 위한 충분한 캐쉬 메모리가 확보되지 않았기 때문에 비디오 캐쉬에 캐쉬되고 있지 않다. 따라서, 새로운 인터벌이 interval_list에서 크기가 가장 작을 때, 비디오 캐쉬에 캐쉬될 수 있는 후보가 된다. I_{im}이 가장 작은 인터벌이고 I_{im}을 위한 충분한 메모리가 free_list에 존재한다면, I_{im}은 비디오 캐쉬에 캐쉬된다. Free_list에 I_{im}을 위한 충분한 메모리가 없다면 cache_list에서 I_{im}보다 큰 인터벌을 victim으로 선택하여 캐쉬 replacement를 수행한다. Cache_list에서 I_{im}보다 큰 인터벌이 없다면, I_{im}은 캐쉬되지 않고, 디스크에서 데이타를 읽어온다. 각 비디오에 대한 스트림 서비스가 종료되

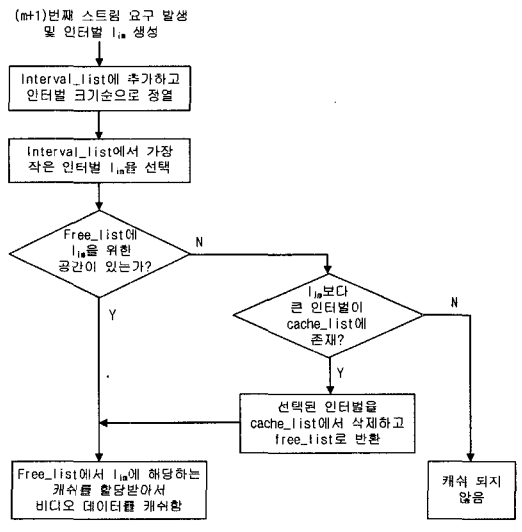

그림 6 인터벌 캐쉬의 캐쉬 정책

면, 비디오 데이타를 캐쉬하는데 사용된 캐쉬 메모리는 free_list로 반환된다. 반환된 캐쉬 메모리가 interval_list에 있는 최소 크기 인터벌을 캐쉬할 만큼 충분한 크기라면, 최소 크기 인터벌은 비디오 캐쉬에 캐쉬된다.

4. 실 험

본 논문에서는 4 대의 Pentium-III 시스템을 Gigabit Ethernet으로 연결한 클러스터 시스템을 사용하여 분산 VOD 서버를 구축하였다. 각 서버는 1.4GHz Pentium-III 프로세서, 512MB 의 메인 메모리, 16GB 의 SCSI 하드디스크 및 66MHz/64bit PCI 버스를 가지고 있으며, 운영체제로는 Linux 커널 2.4.7을 사용하였다. 또한, 각 서버 노드를 연결하기 위한 서버 내부 통신망으로 Intel사의 1000/Pro Gigabit Ethernet 카드와 3Com사의 SuperStack3 4900 Gigabit Ethernet 스위치를 사용하였다. 본 실험에서 사용하는 비디오 데이타는 초당 110KB/s의 평균 전송률을 가지는 MPEG-4 비디오이다.

4.1 M-VIA 및 TCP/IP를 적용한 성능 비교

본 실험에서는 분산 VOD 서버의 내부 통신망을 위한 통신 프로토콜로 M-VIA를 사용했을 때의 성능을 측정하였으며, 비교 대상으로 TCP/IP를 적용한 시스템을 함께 실험하였다. 또한, 본 시스템에 가장 적당한 스트라이핑 블록 크기를 찾기 위해서, 블록크기를 64KB에서 512KB까지 변화시키면서 실험을 하였다. 모든 스트림 요구는 서로 다른 비디오 데이타에 대한 전송 요구로 비디오 캐쉬는 동작하지 않으며, 6초 간격으로 발생한다.

그림 7은 M-VIA와 TCP/IP를 사용했을 때, 분산 VOD 서버가 동시에 서비스 가능한 최대 스트림 수를 보여주며, M-VIA를 사용한 분산 VOD 서버가 TCP/IP를 사용한 경우보다 더 많은 스트림을 사용자에게 서비스함을 알 수 있다. 그리고, 4 노드 VOD 서버를 기준으로 했을 때 M-VIA를 적용한 분산 VOD 서버의 최대 스트림수는 314개로, TCP/IP를 적용한 경우인 282개보다 최대 11.3%의 성능 향상이 이루어짐을 알 수 있다. 이것은 M-VIA 사용으로 인하여 서버 내부 통신망에서 발생하는 통신 오버헤드가 많이 감소하여 데이타 전송시간이 감소했기 때문이다. 또한, 노드수가 증가할수록 TCP/IP에 대한 M-VIA의 성능 향상이 증가하는 것을 볼 수 있다. 이것은 노드수가 증가할수록 분산 VOD 서버의 내부 통신망을 통한 비디오 데이타 전송량이 증가하며, 이와 더불어 M-VIA 사용으로 인한 서버 내부 통신망의 오버헤드가 더욱 감소하기 때문이다.

그림 7을 보면 분산 VOD 서버의 성능은 스트라이핑 블록의 크기가 커질수록 증가한다. 본 논문에서 제안한 분산 VOD 서버에서는 스트라이핑 블록의 크기와 M-

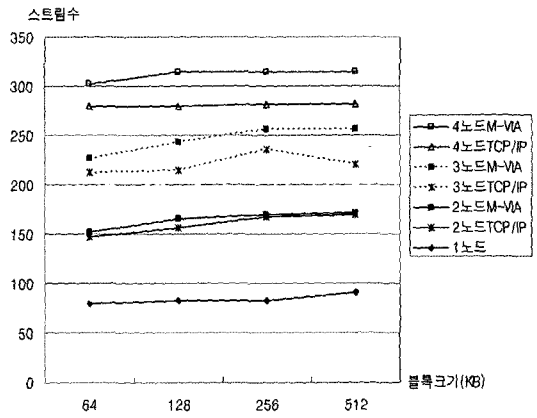

그림 7 M-VIA 및 TCP/IP를 사용한 최대 스트림 수

VIA의 패킷 크기는 동일하다. 따라서, 스트라이핑 블록의 크기가 커질수록 디스크 및 통신망의 대역폭이 증가하고, 이와 더불어 분산 VOD 서버도 성능이 증가한다. 최종적으로 본 논문을 위한 스트라이핑 블록의 크기는 성능 향상이 포화되는 지점인 512KB로 결정하였다.

4.2 인터벌 캐쉬의 성능

본 실험에서는 분산 VOD 서버에 인터벌 캐쉬 기법을 적용했을 때의 성능을 측정하였다. 비디오 캐쉬의 할당량은 50MB, 100MB 및 150MB로 변화시켰으며, 서버 내부 통신 프로토콜은 M-VIA를 사용하였고, 스트라이핑 블록크기는 512KB로 설정하였다. 비디오 캐쉬가 동작하기 위해서는 동일 비디오 데이타에 대한 연속된 스트림 요구가 발생해야 하며, 이를 모델링하기 위해서 각 비디오에 인기도를 부여하였다. 실제 VOD 서비스에서는 인기 비디오에 대한 사용자의 요청 빈도가 더 많을 것이므로 이와 유사한 형태의 사용자 요청을 만들기 위하여 Zipf 의 법칙[25]에 따라 비디오 데이타에 인기도를 부여하였다. 본 실험에서는 20개의 비디오 데이타를 대상으로 하였으며, 각 비디오의 평균 크기는 200MB이다. 각 스트림 요구는 6초 간격으로 발생하도록 하였다.

그림 8과 표 3은 인터벌 캐쉬 기법을 적용했을 때, 분산 VOD 서버가 동시에 서비스 가능한 최대 스트림 수 및 비디오 캐쉬의 hit ratio를 보여준다. 그림 8을 보면 비디오 캐쉬의 크기가 증가할수록 서비스 가능한 스트림 수가 증가하며, 비디오 캐쉬의 크기가 100MB가 되면서 성능향상이 포화된다. 또한, 노드수가 증가할수록 비디오 캐쉬의 성능이 향상되어, 4 노드 분산 VOD 서버를 기준을 했을 때 비디오 캐쉬가 100MB인 경우는 총 420개의 스트림을 서비스하여 382개의 스트림을 서비스하는 비디오 캐쉬를 사용하지 않은 경우보다 약 10%의 성능향상이 발생한다. 이것은 표 3에서 보는 바와 같이 비디오 캐쉬의 크기와 분산 VOD 서버의 노드

수가 증가할수록 비디오 캐쉬의 hit ratio가 증가하여, 원격 디스크에서 비디오 데이타를 전송받는 시간이 감소하기 때문이다. 따라서, 디스크에서 비디오 데이타를 읽는 시간이 감소한 만큼 더 많은 비디오 데이타를 사용자에게 전송할 수 있게 된다.

그림 8을 보면 비디오 캐쉬를 적용했을 때 분산 VOD 서버의 최대 스트림 수는 2, 3, 4 노드 시스템이 1노드 시스템보다 각각 2, 3, 4배 많은 것을 알 수 있다. 이것은 4 노드까지 시스템을 확장했을 때 분산 VOD 서버의 성능이 선형적으로 증가하는 것을 보여주는 것으로, 본 논문에서 제안하는 분산 VOD 서버의 확장성에 무리가 없음을 알 수 있다.

그림 8의 'No cache'와 그림 7의 M-VIA를 4노드를 기준으로 비교해 보자. 그림 8의 경우는 382개로 그림 7에 있는 314개 보다 훨씬 많은 사용자를 서비스하는 것을 알 수 있다. 두 경우는 M-VIA를 사용하고 비디오 캐쉬를 사용하지 않는 점에서 동일하다. 그러나, 그림 8의 실험은 그림 7과 달리 인기도를 부여하여 스트림 요구를 생성했기 때문에, 동일한 비디오 데이타에 대한 연속된 스트림 요구가 발생하고, 디스크에 존재하는 동일 비디오 데이타를 중복적으로 읽게 된다. 이때, Linux의 파일 시스템에서 제공하는 디스크 캐쉬가 동작하여 디스크 접근 시간이 감소하며, 분산 VOD 서버의 성능은 향상된다.

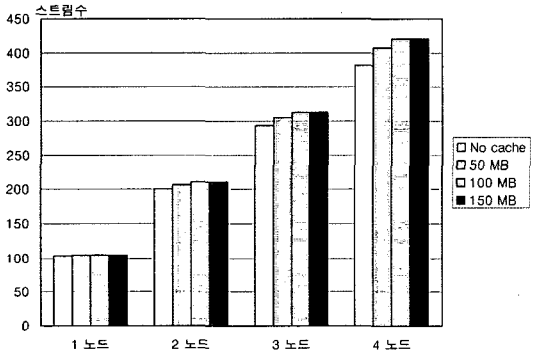

그림 8 인터벌 캐쉬를 적용한 최대 스트림 수

표 3 인터벌 캐쉬 적용에 따른 비디오 캐쉬의 hit ratio

	1 노드	2 노드	3 노드	4 노드
50MB	0.65	0.68	0.68	0.67
100MB	0.76	0.85	0.88	0.89
150MB	0.76	0.85	0.88	0.89

4.3 스트림 요구 간격 변화에 따른 성능

본 논문에서 비디오 캐쉬 알고리즘으로 채택한 인터벌 캐쉬는 인터벌에 해당하는 시간 간격을 기준으로 캐쉬 여부를 결정한다. 따라서, 본 절에서는 스트림 요구의 시간 간격을 3초에서 18초까지 변화시키면서 분산 VOD 서버의 성능을 측정하였다. 노드수는 4 노드를 기준으로 하였다. 본 실험에서 최대 스트림 요구 간격을 18초 결정한 이유는 다음과 같다. 4.2절의 실험에서 4 노드 분산 VOD 서버가 서비스 가능한 스트림수는 420개였다. 비디오 한편의 상영시간이 2시간이고, 2시간동안 420개의 사용자 요구가 균일하게 발생한다고 가정하면, 각 스트림 요구의 평균 시간 간격은 약 18초이다. 이때, 사용자의 스트림 요구가 발생하기 시작한 후 2시간이 지나면, 분산 VOD 서버가 서비스할 수 있는 최대 스트림 수에 도달한다. 이후, 스트림 요구가 평균 18초 간격으로 발생하면, 분산 VOD 서버는 항상 자신이 서비스할 수 있는 최대 스트림을 서비스하게 된다. 또한, 스트림 요구가 18초 보다 작은 경우에도, 분산 VOD 서버는 항상 최대 스트림 수를 서비스한다. 그러나, 스트림 요구가 분산 VOD 서버의 성능을 초과해서 발생함으로, 새로운 사용자 요구는 현재 서비스 중인 사용자 요구가 끝날 때까지 지연된다. 스트림 요구 간격이 18초 보다 크다면, 분산 VOD 서버는 자신의 성능보다 작은 스트림 수를 서비스하게 된다. 따라서, 사용자의 스트림 요구를 최대 18초까지 변화시키면서 실험하였다.

그림 9를 보면 비디오 캐쉬의 크기가 100MB일 때 스트림 요구 간격이 증가하면 분산 VOD 서버의 성능이 감소하는 것을 볼 수 있다. 그러나, 스트림 요구 간격이 9, 12초일 때는 비디오 캐쉬의 크기를 150MB로, 스트림 요구 간격이 18초일때는 비디오 캐쉬의 크기를 200MB로 증가시키면, 분산 VOD 서버가 다시 최대 성능을 발휘하는 것을 알 수 있다. 스트림 요구 간격이 증가하면, 각각의 연속된 스트림에 해당하는 인터벌을 캐쉬하기 위한 캐쉬 요구량은 증가한다. 따라서 비디오 캐쉬의 크기가 고정되어있을 때는 캐쉬할 수 있는 연속된 스트림의 수는 감소한다. 인터벌 캐쉬의 기본 정책은

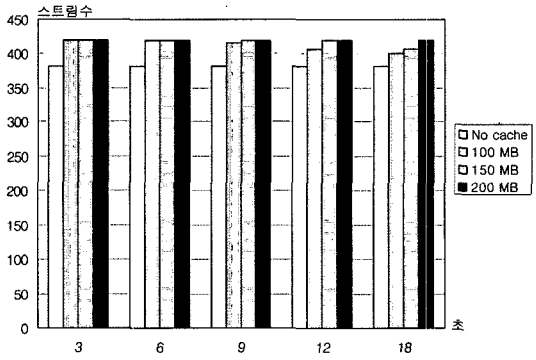

그림 9 스트림 요구 간격 변화에 따른 최대 스트림 수

표 4 스트림 요구 간격 변화에 따른 비디오 캐쉬의 hit ratio

	3초	6초	9초	12초	18초
100MB	0.88	0.89	0.83	0.74	0.6
150MB	0.88	0.89	0.89	0.87	0.74
200MB	0.88	0.89	0.89	0.87	0.87

가능한 많은 수의 연속된 스트림을 캐쉬하여 캐쉬의 hit ratio를 높이는 것이다. 따라서, 본 실험에서는 스트림 요구 간격이 증가함에 따라 표 4와 같이 비디오 캐쉬의 hit ratio가 감소하여 분산 VOD 서버의 성능이 감소한다. 그러나, 비디오 캐쉬의 크기를 200MB까지 확장하면, 비디오 캐쉬의 hit ratio가 다시 증가하여 최대 스트림 수를 서비스하게 된다. 따라서, 20개의 비디오 데이타를 가진 본 실험에서 최적의 비디오 캐쉬 크기는 200MB라고 판단된다.

4.4 서비스 대상 비디오 수의 변화에 따른 성능

4.2와 4.3절의 실험에서는 비디오 캐쉬의 성능을 측정하기 위해서 서비스 대상 비디오로 20개를 사용하였다. 본 절에서는 서비스 대상 비디오 수를 80개까지 변화시키면서 실험을 수행하였다. 스트림 요구 간격은 18초로 하였다.

그림 10에서 보는 바와 같이, 서비스 대상 비디오 수가 증가하면 분산 VOD 서버가 서비스 할 수 있는 최대 스트림 수는 감소한다. 이때, 비디오 캐쉬의 크기를 300MB까지 증가시켜도 분산 VOD 서버의 성능은 더 이상 향상되지 않는 것을 볼 수 있다. 서비스 대상 비디오 수가 증가하면, Zipf 법칙에 따라서 각 비디오에 부여되는 인기도가 감소하며, 비디오 캐쉬의 대상인 동일 비디오에 대한 연속된 사용자 요구가 감소한다. 따라서, 표 5에서 처럼 비디오 캐쉬의 hit ratio가 감소한다. 여기서, 비디오 캐쉬의 크기를 300MB까지 확장하더라도, 비디오 캐쉬의 대상인 연속된 사용자 요구는 증가하지 않기 때문에 비디오 캐쉬의 hit ratio도 증가하지 않으며, 더 이상 분산 VOD 서버의 성능도 증가하지 않는다.

그림 10을 보면, 비디오 파일 수의 증가에 따라서 비디오 캐쉬를 적용한 분산 VOD 서버의 성능도 감소하지만, 비디오 캐쉬를 사용하지 않는 경우에도 분산 VOD 서버의 성능이 감소되는 것을 볼 수 있다. Linux에서는 4.2절에서 설명한 바와 같이 디스크 캐쉬를 지원한다. 비디오 파일 수가 증가하면 동일 비디오에 대한 사용자의 연속된 요구가 감소하고, 디스크에 존재하는 동일 비디오 데이타에 대한 접근 요구도 감소한다. 그리고 디스크 캐쉬의 성능도 떨어지게 된다. 따라서, 비디오 캐쉬를 적용했을 때의 성능을 비디오 캐쉬를 적용하지 않았을 때의 성능과 비교해 보면, 계속 약 10%의 성

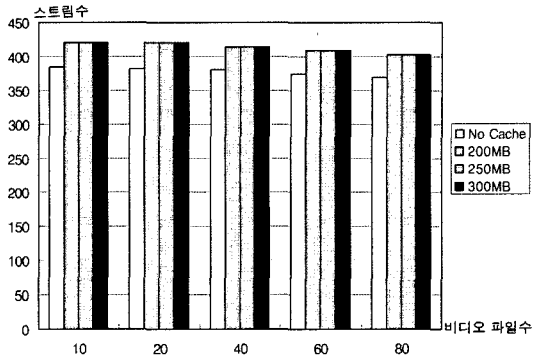

그림 10 서비스 대상 비디오 수의 변화에 따른 최대 스트림 수

표 5 서비스 대상 비디오 수의 변화에 따른 비디오 캐쉬의 hit ratio

	10개	20개	40개	60개	80개
200MB	0.90	0.89	0.85	0.8	0.76,
250MB	0.91	0.89	0.86	0.8	0.76
300MB	0.91	0.89	0.86	0.8	0.76

능 향상을 보이는 것을 알 수 있으며, 서비스 대상 스트림의 수가 증가하더라도 비디오 캐쉬의 효과는 유효할 것으로 판단된다.

실제 서비스에서는 대부분의 사용자들이 보고자 하는 hot video set이 존재하며, 이 hot video set에 대한 사용자 요구 패턴에 의해서 전체 시스템의 성능이 결정될 것이다. 이러한 hot video의 수는 전체 비디오 수에 관계없이 수십 개를 넘지 않으며, 비디오 수가 증가한다고 하여도 각 비디오에 대한 인기도는 변하지 않을 것이다. 반면에 본 절의 실험에서 Zipf distribution에 의해 각 비디오에 부여된 인기도는 비디오수의 증가에 따라서 감소하며, 이것이 비디오 수 증가에 따른 분산 VOD 서버의 성능 감소를 주도하고 있다. 따라서, hot video를 고려한 실제 서비스 상황을 고려해 본다면, 비디오 수 증가로 인하 분산 VOD 서버의 성능 감소는 그림 10에 있는 것보다 작을 것으로 판단된다. 또한, 그림 10에서 80개의 비디오수를 가진 실험은 충분한 수의 hot video를 포함하고 있기 때문에, 비디오의 수가 80이상으로 증가하더라도 분산 VOD 서버의 성능 감소는 크지 않을 것으로 예측된다.

4.3절의 실험에서 스트림 요구간격을 최대 18초로 했을 때 최대 성능을 발휘하는 비디오 캐쉬의 크기는 200MB 였다. 4.4절에서는 서비스 대상 비디오 파일 수를 증가시킬 때, 분산 VOD 서버의 성능은 비디오 캐쉬의 크기가 300MB까지 증가하더라도 변하지 않았다. 따라서, 스트림 요구 간격과 서비스 대상 비디오 파일 수

의 변화를 고려하여 4.3절과 4.4절의 실험결과를 종합했을 때, 본 논문에서 제안하는 분산 VOD 서버가 최대 성능을 발휘하는 비디오 캐쉬의 크기는 약 200MB라고 판단된다.

5. 결 론

분산 VOD 서버에서는 서버 내부 통신망에 발생하는 비디오 데이타의 전송 부하를 최소화하는 것이 중요하다. 본 논문에서는 서버 내부 통신망에 발생하는 부하를 감소시킨 분산 VOD 서버를 제안하였다. 기존의 통신 소프트웨어인 TCP/IP의 오버헤드를 제거하기 위해서 사용자 수준 통신 프로토콜인 VIA를 적용하였으며, 이에 기반한 분산 VOD 파일 시스템을 개발하여 원격 디스크를 사용하는데 소요되는 내부 통신망 비용을 최소화하였다. 그리고, 한번의 send/receive 명령으로 전송 가능한 최대 데이타의 크기를 VOD 서비스에 맞게 확장함으로써 VOD 서버간의 데이타 전송 시간을 감소시켰다. 또한, 원격 서버 노드에서 전송 받은 비디오 데이타를 인터벌 캐쉬 기법을 사용하여 지역 노드의 메인 메모리에 캐쉬함으로써, 서버 내부 통신망에 발생하는 통신량을 감소시켰다. 4노드 시스템을 기준으로 한 실험결과, 분산 VOD 서버의 성능은 노드수가 증가할수록 선형적으로 향상되어 확장성이 좋으며, 스트림 요구 간격 및 서비스 대상 비디오수의 변화에도 꾸준히 성능향상을 보임을 알 수 있었다. 결론적으로 VIA 적용으로 약 11.3%의 성능 향상, 그리고 인터벌 캐쉬 기법을 적용하여 추가로 약 10%의 성능 향상이 생겨 총 21.3%의 성능 향상을 얻을 수 있었다. 또한, 분산 VOD 서버가 최적의 성능을 발휘하기 위해 필요한 비디오 캐쉬의 크기는 200MB로 전체 비디오 데이타의 크기에 비해 상대적으로 작음을 알수 있었다.

참 고 문 헌

[1] Cyrus Shahabi, Roger Zimmermann, Kun Fu, and Shu-Yuen Didi Yao, "Yima: a second-generation continuous media server," *Computer*, Vol.35, Issue 6, pp.56-62, June 2002.

[2] Chang-Soon Park, Mann-Ho Lee, Young-Sung Son, and Oh-Young Kwon, "Design and Implementation of VOD Server by Using Clustered File System," *IEEE International Conference on Multimedia and Expo*, Vol.3, pp.1465-1468, 2000.

[3] M. Barreiro, V. M. Gulias, J. L. Freire, and J. J. Sanchez. "An Erlang-based hierarchical distributed VoD," *7th International Erlang/OTP User Conference(EUC2001)*, Ericsson Utvecklings AB, September 2001.

[4] D. Dunning et al., "The Virtual Interface Architecture," *IEEE Micro*, Vol.18, No.2, pp.66-76, 1998.

[5] A. Dan, D. Dias, R. Mukherjee, D. Sitaram and R. Tewari, "Buffering and Caching in Large-Scale Video Server," *In Proceeding COMPCON. IEEE*, 1995.

[6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W. Su, "Myrinet: A Gigabit-per-second Local Area Network," *IEEE Micro*, Vol.15, No.1, pp.29-36, 1995.

[7] Yuewei Wang, David H. C. Du, "Weighted Striping in Multimedia Servers," *IEEE multimedia systems*, pp.102-109, June 1997.

[8] You-Jung Ahn; Jong-Hoon Kim; Yoo-Hun Won, "A placement policy to maximize the number of concurrent video streams supported in clustered video-on-demand servers," *Proceedings of the IEEE Region 10 Conference*, pp.333-336, Sept. 1999.

[9] K.A. Hua and S.Sheu, "Skyscraper broadcasting: A new boradcasting scheme for metropolitan video-on-demand systems," *In Proc. of ACM SIGCOMM*, Sep 1997.

[10] L-S. Juhn and L-M. Tseng, "Harmonic broadcasting for video-on-demand service," *IEEE Transaction on Broadcasting*, Sept 1997.

[11] Songqing Chen, Bo Shen, Yong Yan; Basu, S., Xiaodong Zhang, "Fast proxy delivery of multiple streaming sessions in shared running buffers," *IEEE Transactions on Multimedia*, Vol.7, Issue 6, pp.1157-1169, 2005.

[12] Guo, M. and Ammar, M.H., "Scalable live video streaming to cooperative clients using time shifting and video patching," *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol.3, pp.1501-1511, 2004.

[13] D. Rotem and J. Zhao, "Buffer management for video database systems," *Proceeding of the international Conference on Data Engineering*, 1995.

[14] B. Ozden, R. Rastogi, and A. Silberschatz, "Buffer Replacement Algorithms for Multimedia Storage Server," *International Conference on Multimedia Computing and Systems*, 1996.

[15] Yinging Zhao and C.-C.Jay Kuo, "Scheduling design for distributed video-on-demand servers," *IEEE International Symposium on Circuits and Systems*, Vol.2, pp.1545-1548, 2005.

[16] Jussara M. Almeida, Derek L. Eager, Mark K. Vernon, "A hybrid caching strategy for streaming media files," *Proceedings of SPIE*, pp.200-212, 2001.

[17] Taeseok Kim, Bahn, H., Koh, K., "Popularity-aware interval caching for multimedia streaming servers," *Electronics Letters*, Vol.39, Issue 21, pp.1555-1557, 2003.

[18] Sarhan, N.J. and Das, C.R., "Caching and scheduling in NAD-based multimedia servers," *IEEE*

Transactions on Parallel and Distributed Systems, Vol.15, Issue 10, pp.921-933, 2004.

[19] Kangho Kim, Jin-Soo Kim, Sung-In Jung, "GNBD/VIA: a network block device over virtual interface architecture on Linux," *Proceedings of International Parallel and Distributed Processing Symposim*, pp.7-13, 2002.

[20] Banikazemi, M., Liu, J., Panda, D.K., Sadayappan, P., "Implementing TreadMarks over Virtual Interface Architecture on Myrinet and gigabit Ethernet: Challenges, design experience, and performance evaluation," *International Conference on Parallel Processing*, pp.167-174, 2001.

[21] P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, and J. Saltz., "Impact of High Performance Sockets on Data Intensive Applications," *Technical Report OSU-CISRC-1/03-TR05*, The Ohio State University, Columbus, OH, January, 2003.

[22] Jae-Wan Jang and Jin-Soo Kim, "Supporting the sockets interface over user-level communication architecture: design issues and performance comparisons," *International Conference on Parallel Processing*, pp.313-320, 2005.

[23] Jack Y.B. Lee, "Parallel Video Servers: A Tutorial," *IEEE Multimedia*, Vol.5, No.2, April/June, 1998.

[24] P. Bozeman and B. Saphir, "A Modular High Performance implementation of the Virtual Interface Architecture," *Proceedings of the 2nd Extreme Linux Workshop*, 1999.

[25] R. L. Axtell, "Zipf Distribution of U.S. Firm Sizes," *Science*, Vol.293, pp.1818-1820, Sept.7, 2001.

오 수 철
정보과학회논문지 : 시스템 및 이론
제 33 권 제 5 호 참조

정 상 화
정보과학회논문지 : 시스템 및 이론
제 33 권 제 5 호 참조