확장성 있는 병렬 I/O VoD 서버의 설계 및 구현

김주훈, 최경희, 김재훈
아주대학교 정보 및 컴퓨터 공학부

Design and Implementation of Scalable Parallel I/O VoD Server

Ju-Hun Kim, Kyung-Hee Choi, Jai-Hoon Kim,
Division of Information & Computer Engineering, Ajou University.

요 약

컴퓨터 하드웨어의 발달로 고속 네트워킹과 대용량 저장 장치가 가능해 지면서 기존의 단순 계산 데이터가 아닌 멀티미디어 데이터를 조작하는 기술이 필요로 하게 되었다. 이 기술을 이용하는 대표적인 것이 VoD 서버 처리 기술이다. 기존 VoD 서버에서는 데이터의 분산 및 버퍼 구조 또는 병렬 I/O 를 통하여 해결하려는 노력이 있어 왔다. 본 논문에서는 고속 네트워킹을 이용하여 확장성을 있는 VoD 서버 모델을 제시한다. 또한 병렬 I/O 에 버퍼 구조를 적용하여 서버의 성능을 극대화 시켰다. 디스크 저장 매체와 서버의 수를 증가시켜 시험하므로써 이 구조가 확장성 있음을 보였다.

1. 서론

고속 통신 및 대용량 저장 장치 등의 컴퓨터 기술의 발달에 따라 예전에는 컴퓨터에서 처리하지 못하던 멀티미디어 데이터의 처리가 가능하게 되었다.

멀티미디어 데이터를 조작하는 기술을 사용하는 대표적인 예가 VoD 서버이다. VoD 서버 설계의 핵심 기술은 (1) 어떻게 네트웍을 구성하여 client 에게 대화형(On Demand)으로 서비스를 할 것인가, (2) 서버에 저장된 데이터를 어떻게 검색, 저장하고 일정한 대역폭으로 서비스할 것인가, (3) 얼마나 경제적으로 구현할 수 있느냐 등으로 생각해 볼 수 있다.

이러한 문제들을 해결하기 위하여 여러 가지 접근 방법들이 연구되었다.. N-VoD 시스템은 많은 사용자 서비스를 목적으로 개발 되었으나 서비스를 위해 특정 사용자는 특정 조건이 이루어 질 때까지 기다려야 하는 제약 사항이 있다[1]. AT&T 에서 개발한 멀티미디어 저장 장치 Fellini server 는 버퍼 구조에 중심을 두어 개발되었다. 그러나 병렬 I/O 를 지원하지 않고 단일 서버구조로 되어 있다[2]. Berkeley 에서 개발된 서버에는 인터넷상에서 데이터를 전송시켜 멀티미디어 데이터 서비스를 제공하는 방법을 채택하고 있다. 즉 단순한 achieve 서버로 구현하였기 때문에 실제 대화형(On Demand)로 서비스하는 데는 문제가 있고 병렬 I/O 를 지원하지 않는다[3].

이러한 노력 중에 병렬 I/O 를 이용한 방법도 있었는데 이러한 병렬 I/O 시스템을 크게 나누어 보면 Proxy-at-Server, Proxy-at-Client, Independent Proxy 로 나눌 수 있다[4]. 이중 IBM 에서 개발한 VoD 서버는 디스크 저장 장치와 Proxy 를 분리하고 고속 네트웍으로 묶는 Independent Proxy 를 채택하고 있다. 그러나 서버내의 버퍼 구조에 대해 고려하지 않았다.

본 논문에서는 Independent Proxy 구조로서 버퍼를 고려한 VoD 서버 모델을 제안한다. 이 모델은 요구되는 성능에 따라 디스크와 서버를 고속 네트워크로 연결할 수 있도록 확장성을 제공한다. 또한 서버내의 버퍼 구조를 적용하여 보다 시스템의 성능을 극대화 할 수 있다.

본 논문의 구성은 다음과 같다 2 장에서는 시스템에 대한 전체적인 모델설명과 버퍼의 구조를 기술하고 3 장에서는 실험방법 및 실험결과를, 4 장에서는 결론 및 향후계획을 기술하였다.

2. 시스템 모델 및 구현

2.1 시스템의 전체 구조

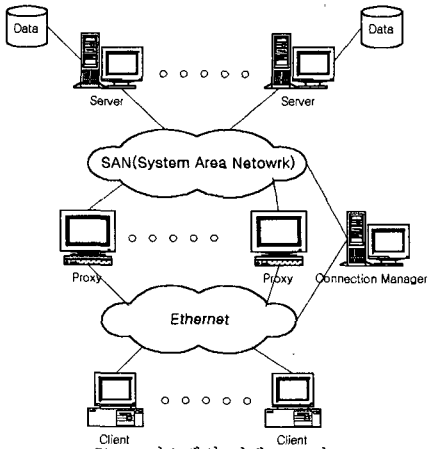

그림 1 시스템의 전체적 개괄

전체적인 시스템의 구조는 그림 1 과 같다. 시스템은 크게 3 가지로 구성된다. VoD 시스템의 전체부하를 분배하고 처리하는 CM(Connection Manager), 클라이언트가 요구한 VoD 데이터를 이더넷(ethernet)을 통하여 전달하는 proxy, 클라이언트(client)에게 전달할 데이터를 읽어 proxy 에 전달하는 server 로

구성되어 진다.

CM 은 client 로부터 영화 데이터 요구를 받고 시스템 구성 요소인 proxy 와 Server 의 부하를 검사한다. 부하 검사를 통해 서비스할 proxy 와 server 를 선택하여 서비스 하도록 요구한다.

Proxy 는 server 에 striping 되어 있는 데이터를 받아 데이터 순서대로 client 에게 전달하는 일을 수행한다. proxy 의 구성은 2 가지의 thread 를 통해 처리되어 진다. 첫번째는 server 에게 필요한 데이터를 요구하고 도착된 데이터를 client 에게 전달하는 client thread, server 로부터 전송 받은 데이터를 해당 버퍼에 저장하는 receiver thread 가 존재한다. (그림 2)

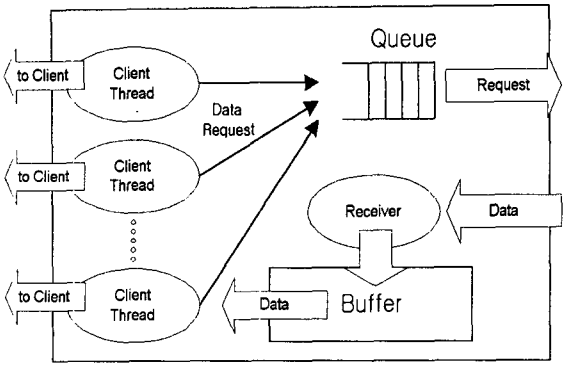

그림 2 Proxy 내부 구조

proxy 는 client 에게 필요한 데이터를 전달하기 위하여 버퍼 구조를 이용한다. 버퍼의 크기가 한정되어 있기 때문에 client thread 는 buffer replace 전략으로 Fellini 의 방법[2]을 수행한다. 즉 현재 서비스하고 있는 client 의 영화상의 순서에서 가장 늦게 읽어질 가능성이 있는 것을 선택하여 buffer replace 를 수행한다.

server 에서는 proxy 의 client thread 로부터 데이터 요청을 받고 해당 데이터를 읽어 데이터를 전달하는 일을 한다(그림 3). 각각의 server 간에는 디스크 배치방법에 따라 proxy 당 서버의 개수가 다르게 될 수 있다. server 내에는 proxy 의 개수 만큼의 process 가 존재하게 된다 server 들간에는 영화 데이터를 striping 시켜 서버간의 load 들 분산시키는 동시에 많은 양의 데이터를 pump 할 수 있는 대역폭을 확보한다. 이 대역폭으로 얻어진 데이터는 SAN(System Area Network)이라는 myrinet 기반의 고속 네트웍을 이용하여 proxy 에게 고속으로 전송 된다. 또한 고속 네트웍으로 server 와 proxy 가 분산되어 있으므로 시스템에서 처리해야 하는 데이터나 동시에 지원해야 하는 client 가 늘어 난 경우 SAN 내에 server 또는 proxy 를 증설할 수 있으므로 확장을 보다 쉽게 할 수 있다.

proxy, server 와 CM 은 PVM(Parallel Virtual Machine) 라이브러리를 이용하여 구축하였기 때문에 통신 메커니즘의 개발 시간을 줄일 수 있고 각각의 노드(컴퓨터)를 하나의 시스템으로 쉽게 연결시킬 수 있다.

2.2 Proxy 의 버퍼 구조

Proxy 내에서의 버퍼 구조는 client thread 와 Receiver thread 가 사용한다.

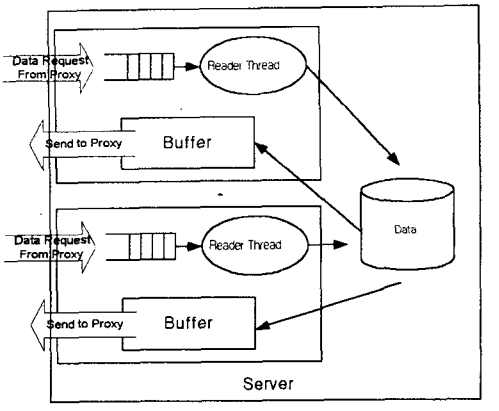

그림 3 Server 내 구조

Client thread 는 client 에게 전송할 데이터가 이미 버퍼 내에 존재하는지 검사를 한다. 만약 존재한다면 그 버퍼의 lock count 를 올려 buffer replace 에 영향을 받지 않게 한다. 존재하지 않는다면 저장할 곳을 할당한 후에 서버에게 데이터를 요청한다. 해당 버퍼에 데이터가 도착할 때까지 기다리고 있다가 데이터를 클라이언트에게 전송한다. Receiver thread 는 client thread 가 요청한 데이터를 server 로부터 받아 해당 버퍼에 데이터를 넣어 주는 일을 한다.

버퍼 내에서는 각 영화별로 영화 clip 들을 관리한다. 모든 영화 clip 은 상영순서에 따라 링크를 구성하고 있다. Client thread 는 데이터를 읽어 낼 때 이 링크를 따라 데이터를 검색한다. 버퍼가 모두 사용 중이어서 버퍼 replace 를 수행할 경우 이 링크를 통하여 영화 clip 의 중요도를 검색한다.

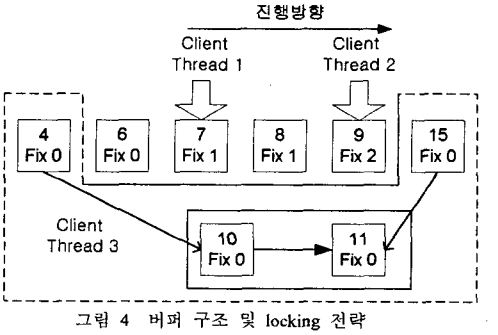

그림 4 버퍼 구조 및 locking 전략

Buffer 할당과 다음 영화 clip 을 검색하기 위해 링크를 따라가다 buffer replace 때문에 검색한 버퍼가 원하던 버퍼가 아닐 수 있다(그림 4).

이러한 현상을 막기 위해서 링크를 읽어 나가는 것과 버퍼 할당하는 것 간의 순서제어가 필요하다. Client thread 는 해당 영화 clip 이 존재하는지를 검색할 때 링크를 읽고, 검색결과 데이터가 없는 경우 버퍼 replace 전략을 수행하고 링크를 바꾼다. Receiver thread 는 server 로부터 받은 데이터를 해당 버퍼에 넣는 일만을 하기 때문에 링크를 읽어 내기만 한다.

즉 상대적으로 링크를 검색하는 일이 링크를 바꾸는 일보다

닫기 때문에 링크에 대한 locking 문제를 Reader's and writer's problem 에서 사용한 방법을 이용하여 해결할 수 있다.

영화 clip 을 저장할 블록을 할당을 했지만 곧바로 데이터가 전달되는 것이 아니기 때문에 client thread 는 각 블록에 대해서 데이터가 도착했는지를 검사하고 있어야 한다. 이것을 receiver thread 가 client thread 에게 알려주는 방법을 통해 데이터도착까지의 데이터 도착 검사 루프를 줄일 수 있다. 즉 서버로부터 전송되는 데이터를 저장할 버퍼에 초기값이 0 인 semaphore 를 주어 해당 버퍼의 데이터를 기다리는 client thread 들을 block 상태로 되게 한다. Receiver thread 가 해당 버퍼에 내용을 채우고 데이터가 도착했음을 해당 버퍼의 semaphore 를 operation 으로 client thread 들에게 알려준다. 해당 block 을 기다리고 있는 client thread 등은 block 에서 풀려나게 되어 바로 데이터 전송에 들어 갈 수 있다.

2.3 Proxy 와 Server 에서의 균등 서비스 전략

Proxy 에서 semaphore 를 적용하였기 때문에 thread 간의 실행 기회가 공정하게 주어지지 않아 Client thread 간의 starvation 이 존재한다. 즉 특정 client thread 는 server 에게 영화 clip 요청을 빈번하게 보내지만 특정 client thread 는 요청을 보내지 못하는 경우가 생긴다.

이것을 해결하기 위해서는 현재 존재하는 thread 간에 기회를 균등하게 주어야 한다. 모든 client thread 가 한번에 한번씩 데이터 clip 을 요청하는 메시지를 전달하도록 하면 된다. 이를 위하여 ticket 알고리즘을 이용하면 된다[6]. 즉 client 의 등록순서를 ticket 으로 설정하여 round-robin 으로 기회를 주면 된다.

server 는 다수의 proxy 에게 영화 clip 을 전송한다. server 는 데이터를 신청하는 proxy 들간에 동등하게 서비스를 해주어야 한다. 그러나 server 에 영화 clip 전송요구를 처리하는 process 를 하나만 두게 되면 병목 현상이 일어나서 서비스 받는 proxy 만 계속하여 서비스 받는 경우가 생긴다. 이것을 막기 위해서 proxy 당 전용 전송 요구를 처리해 주는 process 를 할당한다.

위의 결과는 결국 proxy 와 server 간의 데이터 전송요구와 처리를 하기 위해서는 server 의 load 를 분산하는 방법이 필요한 에가 된다. 특정 서버의 load 가 전체적인 시스템에 성능 저하를 초래 할 수 있기 때문이다. 이것을 위하여 서버의 데이터를 분산시켜 특정서버에 load 가 분산되는 역할이 필요하다. 또한 부하분산을 위하여 같은 데이터를 보유한 서버들간에 어느 서버에 데이터를 요청할 것인가에 대한 연구가 필요하다.

3. 실험 방법 및 실험 결과

실제 client 를 이용하여 테스트 할 수 없기 때문에 proxy 에서 client 로 서비스 하는 것은 고려치 않았다. 영화 한편 정도의 분량을 실험에 사용하였는데 영화 한편을 보기 위해서는 MPEG-I 기준으로 평균 초당 1.5Mbps 를 지원해야 한다. 영화에 대한 server 의 로드를 분산시키고 병렬 I/O 를 수행할 수 있도록 영화데이터를 round-robin 방식으로 striping 하였다. Client 가 거의 동시 요구한 경우 buffering 에서 중복이 많이 되기 때문에 buffer replace 가 간격을 두고 요구하는 것보다 부하가 적어 진다. 때문에 1 초 간격으로 client 를 접속 시켰다.

- 데이터 배치 방법 : 1,638,400byte 씩 round-robin 배치
- 서버당 데이터량 : 총데이터량 640M, 서버가 2 개일 때 320M, 3 개일 때 210M, 4 개일 때 160M
- CM 의 Proxy 와 Server 의 선택 방법: round-robin
- 서비스 성공 판단 기준 평균: 1.5Mbps 의 지원 여부

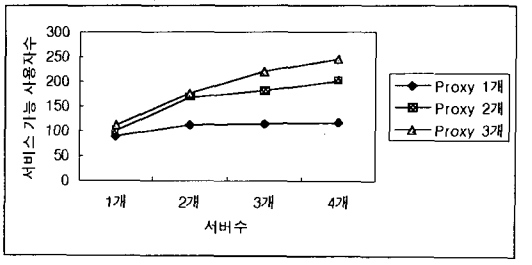

그림 5 실험 결과

실험결과에서 보면 server 또는 proxy 가 증가할 수록 proxy 당 서비스 사용자수가 증가함을 알 수 있다. 이것을 통하여 proxy 와 server 를 증설함으로써 보다 확장성 있게 더 많은 사용자를 서비스 할 수 있음을 알 수 있다.

4. 결론 및 향후 계획

본 논문에서 제시한 VoD 서버가 확장성이 있음을 알 수 있었다. 전체적인 VoD 성능을 높이기 위해서는 proxy 와 server 모든 쪽에서 load 의 분배가 중요한 역할을 한다. 또한 각 proxy 와 server 에 균등하게 서비스를 해주어야 고르게 그리고 보다 많은 사용자에게 서비스 할 수 있음을 알 수 있다. 그러므로 Server 및 Proxy 간의 load 를 분배하는 방법, Striping 방법에 따른 load 의 분배, CM 이 proxy 및 server 를 선택하는 방법 등이 연구되어야 할 것이다.

참고 문헌

[1] A. dan,D. M. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and caching in large-scale video servers", *in Proc. Compcon*, pp. 217-224, March 1995.

[2] B.Ozden,, R. Rastogi and A. Silberschatz, "Buffer Replacement Algorithms for Multimedia Databases," *IEEE International Conference on Multimedia Computing and Systems*, June 1996.

[3] L. A Rowe, D. A. Bergert, and J.E.Baldeschwieler, "The Berkeley Distributed Video-on-Demand System," *Multimedia Computing Proceedings of the sixth NEC Research Symposium*, 1996.

[4] Jack T.B. Lee, "Parallel Video Servers: A tutorial," *IEEE Multimedia* Vol. 5, NO.2 April/June 1998.

[5] Renu Tewari, Rajat Mukherjee, Daniel M. Dias, Harrick M. Vin, " Design and Performance Tradeoffs in Clustered Video Server,"

[6] Gregory R. Andrews, Concurrent Programming, The Benjamin/Cummings Publishing Company, Inc. pp.112-115 pp. 192-197.