제25회 한국정보처리학회 춘계학술발표대회 논문집 제13권 제1호 (2006. 5)

모바일 환경 기반의 소프트웨어 스트리밍 시스템을 위한 선인출 기법의 설계 및 구현

이대우, 박선영, 김진수, 맹승렬
한국과학기술원 전자전산학과
e-mail : dwlee@camars.kaist.ac.kr

Design and Implementation of Prefetching Mechanism for Software Streaming Systems in Mobile Environment

Daewoo Lee, Seon-Yeong Park, Jin-Soo Kim, Seung-Ryoul Maeng
Dept. of Electronic Engineering and Computer Science, KAIST

요 약

온디맨드 소프트웨어 스트리밍(On-Demand Software Streaming)이란 서버가 제공하는 소프트웨어를 클라이언트에 설치하지 않고 실행하는 기술로, 서버로부터 필요한 부분만 스트리밍으로 전송 받아 실행하는 기술을 말한다. 이 기술을 이용하면 소프트웨어 제공자는 소프트웨어 관리를 용이하게 할 수 있고, 소프트웨어 사용자는 적은 저장 공간으로 많은 소프트웨어를 사용할 수 있다는 이점을 얻게 된다. 하지만 모바일 환경에서 이를 이용하는 경우에는, 느린 무선 네트워크를 통해 소프트웨어 이미지를 전송해야 하기 때문에 소프트웨어 실행 속도가 매우 느리다는 문제가 생긴다. 이를 해결하기 위해서 본 논문에서는 온디맨드 소프트웨어 스트리밍을 사용하는 시스템의 성능 향상을 위해 효율적인 선인출 기법을 설계하고 실제로 구현하였다. 실험 결과, 애플리케이션이 데이터를 읽을 때 걸리는 시간이 무선랜 환경에서는 평균 50%, CDMA 환경에서는 평균 20% 정도 감소했으며, 특히 네트워크 지연시간이 증가할수록 더 많이 감소하였다.

1. 서론

최근 모바일 단말에서 사용자들이 필요에 따라 사용할 수 있도록 소프트웨어를 제공하는 방법으로 온디맨드 소프트웨어 스트리밍(On-Demand Software Streaming)이 각광을 받고 있다[1]. 온디맨드 소프트웨어 스트리밍은 서버가 제공하는 소프트웨어를 클라이언트에 설치하지 않고 실행하는 기술로, 서버로부터 필요한 부분만 스트리밍으로 전송 받아 실행하는 기술을 말한다. 이 기술을 이용하면 소프트웨어 제공자는 소프트웨어 관리에 관련된 여러 가지 문제들(새로운 기능 추가, 버그 수정 등)을 간단하게 해결할 수 있고, 소프트웨어 사용자는 적은 저장 공간으로 많은 소프트웨어를 사용할 수 있다는 이점을 얻게 된다. 하지만 모바일 환경에서 이를 이용하는 경우에는 느린 무선 네트워크를 통해 소프트웨어 이미지를 전송해야 하기 때문에 소프트웨어 실행 속도가 매우 느리다는 단점이 있다. 이러한 문제를 해결하기 위해서는 애플리케이션이 어떤 블록을 요청할 것인지를 예측하여 미리 서버로부터 받아오는 선인출이 필요하다.

본 연구는 온디맨드 소프트웨어 스트리밍을 사용하는 시스템의 성능 향상을 위해 효율적인 선인출 기법을 설계하고 실제로 구현하는 것을 목적으로 한다. 실험 결과, 애플리케이션이 데이터를 읽을 때 걸리는 시간이 무선랜 환경에서는 평균 50%, CDMA 환경에서는 평균 20% 정도 감소하였다. 특히 네트워크 지연시간이 증가할수록 더 많이 감소하는 것을 확인할 수 있었다.

2. 관련 연구

온디맨드 소프트웨어 스트리밍은 소프트웨어 전체를 다운로드 받아서 설치해야 하는 번거로움을 피하기 위한 점진적 소프트웨어 전송(Incremental Software Delivery) 기법에 영향을 받았다. 점진적 소프트웨어 전송 기법에 대한 연구들로는 유려한 소프트웨어

(Spontaneous Software)[7], 스트리밍 모듈(Streaming Module)[6] 등이 있다. 유려한 소프트웨어를 실행하는 경우에는 실행 중에 어떤 파일이 필요하게 될 때 서버로부터 전송 받는다. 스트리밍 모듈을 실행하는 경우에는 가장 먼저 실행되는 모듈을 서버로부터 전송 받고 나머지는 스트리밍 스터브 프로시저(Streaming Stub Procedure)로 대체해서 받는다. 그리고 백그라운드 작업으로 나머지 모듈을 전송 받는다. 이들은 실행 도중에 다운로드 받아야 하는 데이터가 큰 경우에 사용자가 기다려야 하는 시간이 지나치게 늘어나는 단점이 있다. 블록 스트리밍(Block Streaming)[4]은 소프트웨어를 일정 크기의 블록으로 나눠서 스트리밍을 하지만, 블록으로 나누는 과정에서 기존의 소프트웨어를 약간 수정해야 한다.

3. 시스템 설계

선인출 기법을 적용한 소프트웨어 스트리밍 시스템을 설계하기 위해서 다음과 같이 네 가지를 고려하였다. 첫째, 자원 제약이 큰 클라이언트에서 선인출에 의한 오버헤드가 최소화되도록 설계하였다. 둘째, 진행 중인 클라이언트의 데이터 요청 처리가 선인출 때문에 지연되지 않도록 설계하였다. 셋째, 기존의 소프트웨어를 추가적인 노력 없이 사용할 수 있도록 설계하였다. 넷째, 사용자의 소프트웨어 사용 패턴에 동적으로 빠르게 적응할 수 있도록 설계하였다. 그림 1 은 이와 같은 요구 조건을 만족하도록 설계한 소프트웨어 스트리밍 시스템의 구조를 나타낸다.

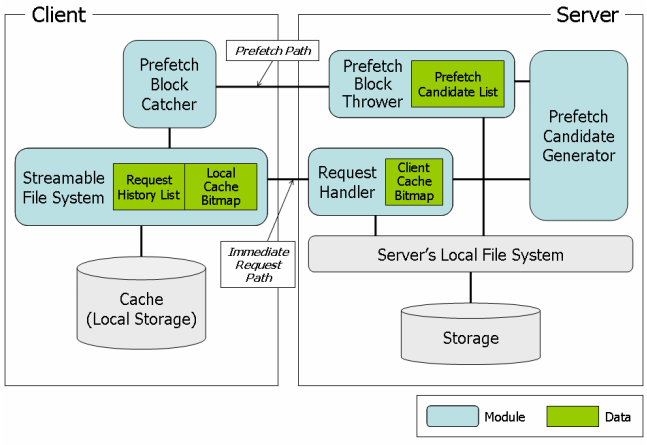

그림 1. 소프트웨어 스트리밍 시스템 구조

클라이언트는 서버가 제공하는 소프트웨어 이미지를 파일 시스템 레벨에서 온디맨드 소프트웨어 스트리밍으로 전송 받아 실행한다. 기본적으로 모든 소프트웨어 이미지는 서버에만 존재하며, 클라이언트의 로컬 저장장치(local storage)는 클라이언트와 서버 간의 캐시(cache)로 사용한다. *스트리머블 파일 시스템(Streamable File System)*은 클라이언트의 파일 시스템으로써, 애플리케이션이 데이터를 읽기 위한 블록 요청을 보내면 그에 해당하는 데이터를 캐시로부터 읽거나 *긴급 요청 경로(Immediate Request Path)*를 통해 서버로부터 받아온다. 이 때, 과거의 블록 요청을 기록해놓은 *요청 내역 목록(Request History List)*의 정보를 서버에게 보내서, 과거 기록을 선인출에 반영할 수 있도록 한다. 그리고 서버로부터 블록을 받을 때마다 *로컬 캐시 비트맵(Local Cache Bitmap)*을 갱신하여 어떤 블록이 캐시에 저장되어 있는지를 유지한다. *선인출 블록 수집기(Prefetch Block Catcher)*는 *선인출 경로(Prefetch Path)*를 통해서 서버가 보낸 선인출 블록들을 받는다. 긴급 요청 경로를 통해 받으면 선인출로 인해 클라이언트의 데이터 요청 처리가 지연될 수 있기 때문이다.

서버는 클라이언트가 요청한 소프트웨어 이미지를 전송하며, 클라이언트의 사용 패턴을 분석하여 가까운 미래에 요청할 블록을 예측하고 선인출한다. *요청 처리기(Request Handler)*는 클라이언트가 요청한 블록을 디스크에서 읽어서 전송한다. 이 때 받은 클라이언트의 과거 기록을 *선인출 후보 생성기(Prefetch Candidate Generator)*에게 전달하여 선인출에 반영하도록 한다. 선인출 후보 생성기는 클라이언트가 가까운 미래에 요청할 블록들을 예측하여 *선인출 후보 목록 (Prefetch Candidate List)*에 반영하며, *선인출 블록 전송기 (Prefetch Block Thrower)*는 이들을 디스크에서 읽어서 선인출 경로를 통해 클라이언트에게 전달한다. *클라이언트 캐시 비트맵(Client Cache Bitmap)*은 클라이언트가 어떤 블록을 가지고 있는지를 기록한 것으로, 클라이언트에게 블록을 전송한 다음에는 항상 갱신한다. 따라서 이를 초기화하기 위해서는 클라이언트가 소프트웨어의 실행을 요청할 때 로컬 캐시 비트맵을 전송하는 것이 필요하다.

선인출 후보 생성기는 서버의 핵심 모듈로써, 클라이언트의 과거 기록을 지속적으로 수집하면서 선인출 알고리즘을 수행한다. 이는 사용자의 소프트웨어 사용 패턴에 빠르게 적응하며 가까운 미래에 요청될 블록을 예측하여 효율적으로 선인출하기 위해 설계하였다. 또한 기존의 소프트웨어를 추가적인 노력 없이 사용할 수 있도록 과거 사용 기록을 기반으로 하는 선인출 알고리즘인 PPM(Prediction by Partial Match-ing)[2][3]을 이용하였다. 이와 같은 처리 과정은 많은 자원을 필요로 하기 때문에 클라이언트에 비해 자원 제약이 적은 서버에 배치하였다.

4. 동작 시나리오

본 절에서는 소프트웨어가 실행될 때 클라이언트와 서버 간에 데이터가 송수신되는 과정 및 선인출되는 과정을 설명한다.

4.1. 기본적인 동작

애플리케이션이 요청한 블록이 클라이언트의 캐시에 없으면 그림 2 와 같은 과정이 진행된다. 스트리머블 파일 시스템이 애플리케이션의 블록 요청을 받으면 ① 로컬 캐시 비트맵을 확인하여 그 블록이 캐시

에 있는지 확인한다. 만약 없다면 ② 요청 내역 목록을 읽어서 ③ 긴급 요청 경로를 통해 서버의 요청 처리기에게 현재의 블록 요청과 요청 내역 목록을 보낸다. ④ 요청 처리기는 클라이언트 캐시 비트맵을 참조해서 서버가 그 블록을 보낸 적이 있는지 확인하고 ⑤ 없다면 요청 내역 목록을 선인출 후보 생성기에게 전달해서 선인출 알고리즘에 반영하도록 한다. ⑥ 요청 처리기는 요청 받은 블록을 디스크에서 읽어서 다시 스트리머블 파일 시스템에게 전송하고, ⑦s 클라이언트 캐시 비트맵에 요청 받은 블록을 보냈다고 기록한다. 스트리머블 파일 시스템은 받은 블록을 캐시에 저장하고 ⑦c 로컬 캐시 비트맵에 기록한 다음, ⑧ 요청 내역 목록을 갱신하고 나서 블록을 애플리케이션에게 리턴한다.

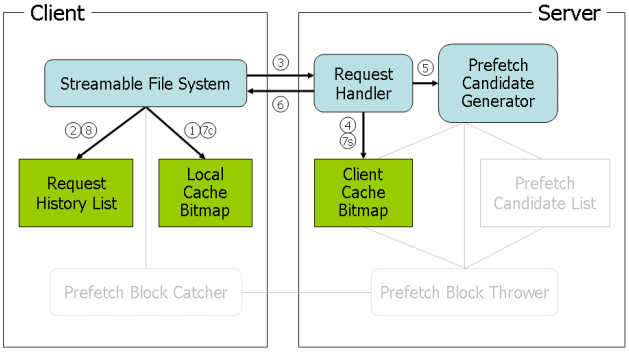

그림 2. 블록 요청에 대한 기본적인 처리 과정

⑤에서 선인출 후보 생성기가 요청 내역 목록을 받으면 그림 3 과 같이 선인출이 시작된다. 기본적인 처리 과정과 동시에 진행되지만, 우선 순위가 낮기 때문에 기본적인 처리 과정을 방해하지 않는다.

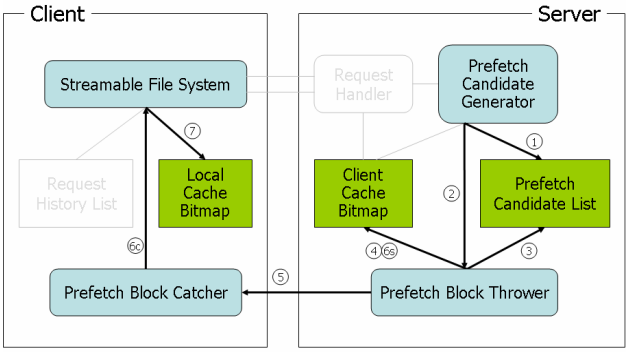

그림 3. 기본적인 선인출 과정

① 선인출 후보 생성기는 선인출 알고리즘을 수행해서 선정된 선인출 후보 블록을 선인출 후보 목록에 등록한다. 이 때, 선인출 후보 목록이 비어있지 않다면 비운다. ② 그리고 선인출 블록 전송기에게 선인출 후보 목록이 갱신되었음을 알린다. ③ 선인출 블록 전송기는 선인출 후보 목록에 등록된 블록을 디스크에서 읽은 다음, ④ 클라이언트 캐시 비트맵을 확인하여 클라이언트에게 이미 전송했던 블록이 아님이 확인되면 ⑤ 선인출 경로를 통해 클라이언트의 선인출 블록 수집기에게 전달한다. ⑥s 그리고 클라이언트 캐시 비트맵을 갱신한다. ⑥c 선인출 블록 수집기가 받은 블록은 스트리머블 파일 시스템이 저장하고, ⑦ 로컬 캐시 비트맵을 갱신한다. 선인출 후보 목록이 비어있지 않다면 ③부터 ⑦까지의 과정이 반복된다.

4.2. 선인출의 적시를 놓친 경우

어떤 블록에 대한 선인출이 끝나기 전에 애플리케이션이 그 블록을 요청하는 경우가 생길 수가 있다. 이러한 경우를 *선인출의 적시를 놓침(untimely prefetch)* 경우라고 정의한다. 스트리머블 파일 시스템은 이에 대한 정보를 알 수 없기 때문에 그 블록을 서버에 요청하게 되므로, 같은 블록을 두 번 전송하게 되어서 대역폭을 낭비하게 된다. 그림 4 는 이와 같은 문제를 해결하기 위한 시나리오이다.

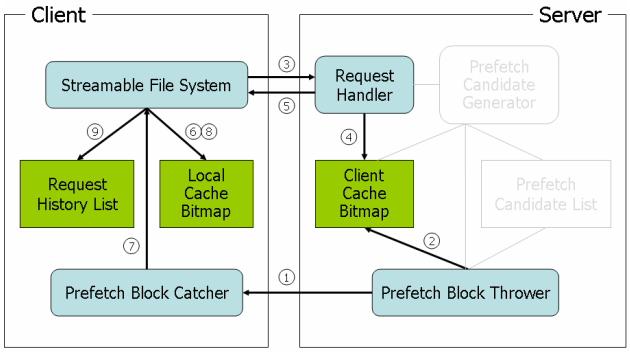

그림 4. 선인출의 적시를 놓친 경우의 처리 과정

① 먼저 선인출 블록 전송기가 블록을 전송한 다음에 ② 클라이언트 캐시 비트맵에 블록을 전송했다고 기록한다. ③④ 그 후, 요청 처리기가 그 블록에 대한 블록 요청을 받으면, ⑤ 요청 처리기는 스트리머블 파일 시스템에게 요청한 블록이 선인출 중이니 끝날 때까지 기다렸다가 캐시에서 읽으라는 메시지를 보낸다. ⑥ 그러면 스트리머블 파일 시스템은 로컬 캐시 비트맵을 확인해서 그 블록이 캐시에 있는지, 즉, 선인출이 끝났는지를 확인한다. 왜냐하면 긴급 요청 경로를 통해 메시지를 주고 받는 동안 선인출이 끝날 수도 있기 때문이다. 만약 선인출이 끝나지 않았다면 슬립(sleep) 상태로 전환한다. ⑦ 선인출 블록 수집기가 선인출된 블록을 받아서 스트리머블 파일 시스템을 깨우면(wakeup), ⑧ 스트리머블 파일 시스템은 그 블록을 캐시에 저장한다. ⑨ 그리고 요청 내역 목록을 갱신한 다음, 블록을 애플리케이션에게 리턴한다.

5. 실험 결과

우리가 설계한 시스템을 실제로 구현하여 다양한 실험을 통해 성능을 평가하였다. 실험 환경은 다음과 같다. 모바일 단말로는 Hybus X-Hyper270A 보드를 이용하였다. Hybus X-Hyper270A 보드는 Intel PXA270 프로세서(520MHz)를 탑재하고 있는 고성능 임베디드 시

스템 설계 보드로, 64MB RAM 과 10Mbps 이더넷(Ethernet)을 내장하고 있다. 또한 USB 1.1 을 제공하고 있기 때문에 로컬 저장장치로 1GB 의 USB 메모리를 이용하였다. 서버로는 Pentium IV 2.8GHz 와 1GB 의 램을 내장한 일반 PC 를 사용하였다. 그리고 Pentium IV 2.4GHz 와 512MB RAM 및 100MB 이더넷을 내장하고 있는 PC 에 NISTNet network emulator[5]를 설치하여 클라이언트와 서버 사이의 네트워크 환경을 다양하게 시뮬레이션 하였다. 실험 결과로는 클라이언트의 캐시를 비워놓은 상태에서 세 번의 동일한 실험을 수행하여 얻은 평균값을 이용했으며, 서버의 파일 시스템의 버퍼 캐쉬(buffer cache)에 의한 영향을 줄이기 위해서 첫 번째 실험 결과는 버렸다.

실험에 사용된 소프트웨어들은 doom 과 pdfviewer 로, 사용자 입력을 기다리는 대화식 소프트웨어(interactive software)이다. 따라서 소프트웨어의 전체 수행 시간을 측정하는 것은 의미가 없기 때문에 성능 향상을 판단하기 위한 척도로써 평균 readpage 시간을 측정하였다. *평균 readpage 시간*이란 스트리머블 파일 시스템이 캐시나 서버로부터 블록 하나를 읽을 때 소요된 평균 시간을 의미하며, 따라서 이는 소프트웨어 스트리밍의 성능을 수치적으로 나타낸 것이라고 할 수 있다.

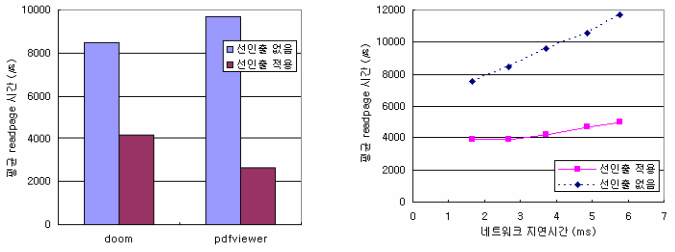

그림 5. 무선랜 환경을 시뮬레이션한 결과

그림 5 는 무선랜(wireless LAN) 환경을 시뮬레이션 했을 때의 실험 결과이다. 왼쪽 그림은 선인출 기법을 적용했을 때 doom 과 pdfviewer 에 대한 성능 향상을 나타낸다. 선인출 기법을 적용한 경우에 그렇지 않은 경우보다 평균 readpage 시간이 50% 이상 감소했다. 오른쪽 그림은 같은 실험 환경에서 네트워크 지연시간을 변화시켰을 때의 평균 readpage 시간을 측정한 것이며, doom 과 pdfviewer 모두 비슷한 현상을 보였다. 선인출 기법을 적용하면 네트워크 지연시간이 증가할 때 선인출을 하지 않는 경우보다 평균 readpage 시간이 더 적은 비율로 증가하며, 또한 성능이 향상된 폭도 더 늘어나는 것을 확인할 수 있다. 이는 애플리케이션이 블록을 요청했을 때 서버로부터 받아와야 하는 경우가 줄어들면서 네트워크 상태의 영향이 적게 나타나기 때문이다.

그림 6 은 CDMA(Code Division Multiple Access) 환경을 시뮬레이션 했을 때 대역폭과 네트워크 지연시간의 변화에 따른 pdfviewer 의 평균 readpage 시간의 변화를 나타낸 그림이다. 앞선 실험과 마찬가지로 선인출 알고리즘으로는 PPM 을 사용했다. 전체적으로 무선랜 환경에서는 50% 이상의 이득을 얻을 수 있었지만, CDMA 환경에서는 20% 정도로 감소한 것을 알 수 있다. 또한 대역폭이 부족할수록 이득 또한 줄어들었지만, 네트워크 지연시간에 대한 영향은 여전히 적다는 것도 알 수 있다.

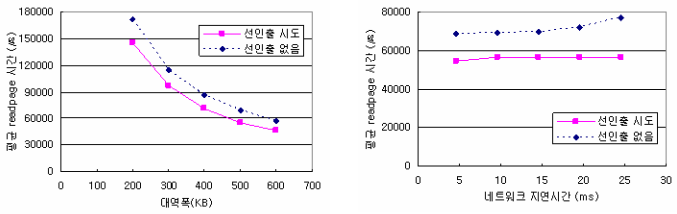

그림 6. CDMA 환경을 시뮬레이션한 결과

6. 결론

느린 네트워크를 기반으로 하는 모바일 환경에서 온디맨드 소프트웨어 스트리밍을 효율적으로 하기 위해서 우리는 선인출 기법을 적용한 소프트웨어 스트리밍 시스템을 설계하였고, 이를 실제로 임베디드 시스템 설계 보드 상에서 구현하였다. 실험 결과, 선인출 기법을 통해서 50% 이상의 성능이 향상되었고, 느린 CDMA 환경에서도 20% 정도 성능이 향상되었다. 향후 다양한 실험을 통해서 소프트웨어 스트리밍 시스템에 적합한 선인출 알고리즘의 특성을 분석하여 보다 효율적인 선인출 기법을 개발할 계획이다.

참고문헌

[1] 최완, 허성진, 김원영, 김준, 남기혁, 김명준, 송동호, 박세영, "온디맨드 소프트웨어 스트리밍 기술현황 및 개발방향," *전자통신동향분석*, 2004 년.

[2] T. C. Bell, J. C. Cleary, and I. H. Written, *Text Compression*, Prentice-Hall Advanced Reference Series, 1990.

[3] T. M. Kroeger and D. D. E. Long, "Design and Implementation of a Predictive File Prefetching Algorithm," *USENIX Annual Technical Conference*, Jun. 2001.

[4] P. Kuacharoen, V. J. Mooney and V. K. Madisetti, "Software Streaming via Block Streaming," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Sep. 2003.

[5] NIST Net Network Emulator, http://www.itl.nist.gov/div892/itg/carson/nistnet

[6] U. Raz, Y. Volk, and S. Melamed, "Streaming Modules," *U.S. Patent 6,311,221*, Oct. 2001.

[7] G. E. Silveira, "Spontaneous Software: A Web-based, Object Computing Paradigm," International Conference on Software Engineering (ICSE), Jun. 2000.