NSS(Network Streaming Service) 상에서의 Prefetching Algorithm 설계

김영만*, 한왕원*, 허성진**, 최완**
*국민대학교 컴퓨터학부 통신실험실, **한국전자통신연구원
ymkim@cclab.kookmin.ac.kr, cronoss@hitel.net, sjheo@etri.re.kr, wchoi@etri.re.kr

Design of the Prefetching Algorithm in the NSS

Young Man Kim*, Wang Won Han* , SeongJin Heo**, Wan Choi**
*School of Computer Science, Kookmin University,
**Electronics and Telecommunications Research Institute
ymkim@cclab.kookmin.ac.kr, cronoss@hitel.net, sjheo@etri.re.kr, wchoi@etri.re.kr

요 약

본 논문에서는 클라이언트가 NSS서버에서 제공하는 패키지를 네트워크 스트리밍을 통하여 실시간으로 이용 가능하게 해주는 리눅스[1]기반의 NSS(Network Streaming Service) 시스템 상에서 패키지 prefetching 기능을 제공하는 부분에 대한 효율적인 알고리즘의 설계를 다룬다.

1. 서 론

NSS(Network Streaming Service)는 리눅스상에서 클라이언트가 네트워크의 streaming service를 통하여 server측의 패키지를 자신의 local disk상에 설치된 것처럼 실시간으로 이용 가능하게 해주는 시스템이다. 이미 온디맨드방식을 통하여 중앙 서버의 응용SW를 windows환경의 클라이언트에 송부해 사용 가능케 하는 소프트온넷사의 Z!stream[2]이 국내에 알려져 있다. 이 스트리밍방식의 SW는 중앙 서버에서 패키지를 통합·관리하기 때문에, 클라이언트의 패키지 설치 및 유지보수 시간을 효과적으로 줄일 수 있으며 스트리밍을 통하여 다양한 소프트웨어 컨텐츠를 사용할 수 있다. NSS 시스템은 성능을 높이고 네트워크자원을 보다 효율적으로 사용하기 위해 패키지 prefetching이 사용될 수 있는데 prefetching은 네트워크 자원이 사용되고 있지 않은 시간에 클라이언트가 다음에 사용할 가능성이 높은 페이지를 미리 서버에서 읽어와 클라이언트의 로컬디스크에 저장하는 기능이다. 클라이언트는 페이지 요구 시 우선 로컬디스크를 검사하고 미리 로컬디스크로 읽어온 페이지중에 원하는 페이지가 존재한다면 네트워크를 통해서 페이지를 읽지 않고 로컬디스크에서 직접 읽어오므로 NSS의 성능을 개선할 수 있다. 본 논문에서는 NSS 시스템 프로세스를 분석하여 패키지 스트리밍에 적합한 prefetching algorithm을 설계하고자 한다.

2. Prefetching algorithm 설계

본 절에서는 NSS의 disk caching부터 시작하여 prefetching에 관련된 몇 가지 기능들을 설명한다.

2.1 Disk caching

Disk caching은 서버의 로컬디스크로부터 읽어온 페이지를 클라이언트의 로컬디스크의 캐쉬 파티션에 저장하여 금후 같은 페이지를 요구할 때 네트워크를 통해서 서버에서 다시 읽어오는 것이 아니라 클라이언트의 local disk의 캐쉬 파티션에서 읽어오는 기능이다[3]. 이렇게 함으로써 동일한 페이지의 첫번째 접근 이후에는 네트워크를 통한 접근이 아니라 클라이언트 자신의 로컬디스크에서 읽어오므로 네트워크의 전송지연이 생략되어 첫번째 접근시에

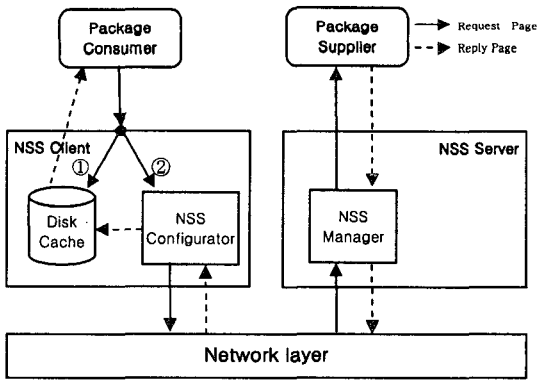

그림 1 NSS상의 diskcaching 실행절차

비해서 접근속도가 훨씬 빨라지게 된다. 다시말해서, 디스크가 가지는 영구적인 저장능력으로 인하여 프로그램 재실행시나 재부팅시에도 네트워크의 참조가 생략되는 장점이 있다. [그림 1]은 disk caching기능의 실행흐름을 나타내고 있다. 클라이언트의 package consumer에서 페이지 부재가 발생하여 해당페이지를 읽을 수 없다면 NSS 시스템이 해당 페이지를 제공해주어야 한다. NSS 시스템은 package consumer가 원하는 페이지가 local disk의 캐쉬 파티션에 존재하는 경우에는 캐쉬 파티션에서 바로 읽어오지만 원하는 페이지가 캐쉬 파티션에 존재하지 않으면 NSM(Network Streaming Module)을 통해서 package supplier로부터 읽어와 클라이언트의 캐쉬파티션에 저장하고 package consumer에게 제공하게 된다. 그 이후 동일한 페이지를 접근하게 되면 캐쉬파티션에 이미 페이지가 저장되어져 있으므로 서버로부터 페이지를 읽어올 필요가 없게 된다.

2.2 Fill-in demon

서버로부터 읽어온 페이지를 클라이언트의 로컬디스크상에 저장하는 disk caching외에 클라이언트가 다음에 읽을 페이지를 미리 저장하는 방법으로 패키지 실행에 대한 성능향상을 가져올 수 있다. 즉 클라이언트가 페이지를 요구할 때에만 전송하는 것이 아니라 클라이언트가 다음에 어떤 페이지를 사용할지 예측해서 클라이언트의 요구가 있기전에 미리 로컬 디스크 캐쉬에 저장한다면 캐쉬 적중률을 좀더 높일 수 있고 클라이언트는 보다 빠른 속도로 패키지를 실행할 수 있을 것이다. 이러한 방법들 중 하나는 클라이언트가 로컬디스크 캐쉬의 비트맵 정보를 이용하여 NSS configurator가 쉬고 있을 때 테이블에 유효 비트가 0인 페이지(로컬디스크 캐쉬에 저장되어 있지 않는 페이지)에 대해 서버에 전송 요청하여 미리 저장하는 것이다.

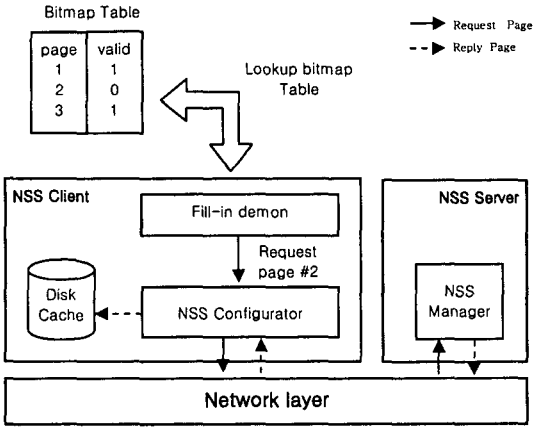

그림 2 fill-in demon 실행

[그림 2]는 fill-in demon의 동작과정을 보여주고 있는데fill-in demon은 로컬디스크 캐쉬의 비트맵 정보를 토대로 현재 유효 비트가 0인 page들에 대해 NSS configurator에 이 페이지의 전송을 요구한다. 해당 페이지가 전송되면 비트맵 테이블의 유효 비트를 1로 설정하여 전송된 페이지라는 것을 표시한다. 이 비트맵 테이블은 로컬 디스크의 캐쉬 파일마다 하나씩 존재하며 클라이언트의 메모리에는 캐쉬 파일들의 비트맵 정보를 가지고 있으면서 모든 page의 유효 비트가 설정된 비트맵 테이블은 완료 체크를 하여 해당 캐쉬 파일의 비트맵 테이블을 메모리에서 제거한다. [그림 3]은 이에 대한 설명이다. 이 그림에서는 전체 비트맵 테이블의 캐쉬 파일 2번에 대한 유효 비트가 체크 되고 prefetching이 완료 되는 순간 해당비트맵 테이블을 메모리에서 제거 시킨다.

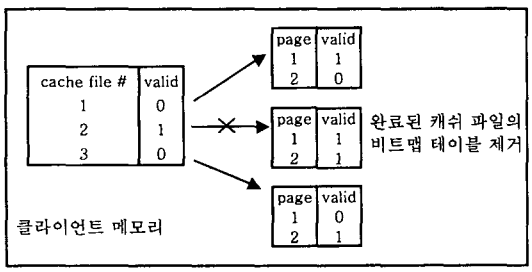

그림 3 클라이언트에서의 비트맵 테이블 관리

위와 같은 방법을 사용함으로써 클라이언트는 페이지 요구가 생길 때마다 네트워크를 통하여 페이지를 제공 받는 것보다 미리 저장 되어진 로컬 디스크 캐쉬파일에서 데이터를 읽음으로써 신속한 패키지 실행속도를 보장받는다.

2.3 Dynamic page ordering 방법

Package consumer가 요구하지 않은 페이지를 미리 package supplier로부터 읽어와 클라이언트의 캐쉬파티션에 저장하는 작업을 담당하는 fill-in demon은 캐쉬파일에 저장되지 않은 page를 순차적으로 서버로부터 prefetching하게 된다. 하지만 이렇게 읽어온 페이지가 클라이언트에서 사용되지 않는다면 성능개선이 안되므로 이러한 문제점을 개선하기 위해 곧 사용되어질 가능성이 높은 페이지를 우선적으로 저장하는 스케쥴링 방법이 필요하다. Dynamic page ordering 방법은 패키지내의 페이지 prefetching의 우선순위로 예전의 여러 클라이언트들에서 발생한 페이지 접근이벤트들을 사용한다. 클라이언트가 서버에게 페이지 요구를 할 때마다 페이지들의 우선순위를 관리하는 테이블의 정보는 갱신된다. [그림4,5]는 Dynamic page ordering 방법을 설명하고 있다.

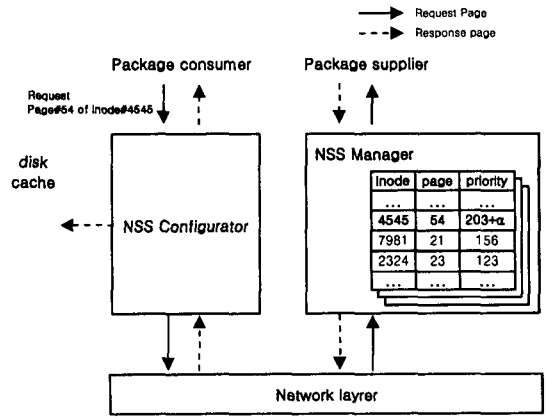

그림4 priority table 의 갱신

[그림 4]는 priority table이 갱신되는 예를 보여주고 있다. Package consumer가 inode 번호가 4545인 파일의 54번째 페이지를 NSS configurator에게 요구하면, 그 요청은 서버의 NSS manager까지 전달되게 된다. NSS Manager은 클라이언트가 요구한 페이지를 supplier로부터 받아서 NSS Configurator에게 전달하고, 스트리밍 서비스되고 있는 해당 패키지의 priority table에서 inode번호가 4545이고 페이지번호가 54인 엔트리를 찾아서 우선순위를 높여주게 된다.

[그림 5]는 다운로드 받은 *priority* table을 사용해서 fill-in demon이 서버에게 페이지를 요구하는 과정을 보여주는 그림이다. Fill-in demon은 prefetching할 페이지를 선택하기 위해서 priority table을 검색하는데, 우선순위가 가장 높은 inode #4545, page #54가 이미 로컬디스크에 저장되어 있으므로 fill-in demon은 다음으로 우선순위가 높고, 아직 로컬디스크에 저장되지 않은 inode#4545, page #55를 선택하고 NSS configurator에게 이 페이지를 요구한다. 요청을 받은 NSS Configurator는 서버로부터 페이지를 읽어와 캐쉬 파티션에 저장한다.

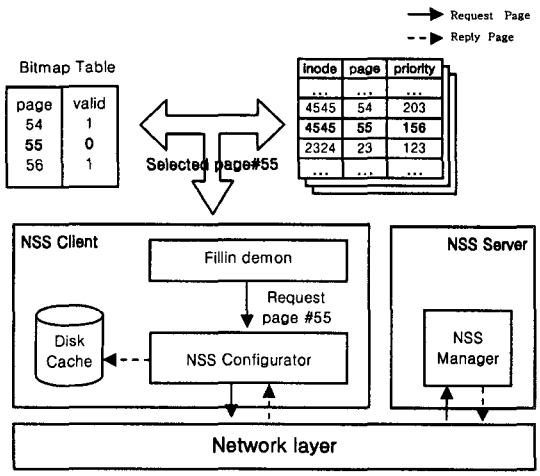

그림5 Priority Table를 이용한 페이지 요구

Dynamic page ordering 방법에서 priority table의 정확도를 높이기 위해서 package consumer와 fill-in demon의 페이지 요구를 구별해줄 필요가 있다. Priority table은 여러 클라이언트들의 page 사용요구들을 누적하여 작성하기 때문에 NSS 시스템가동 초기의 priority table은 신뢰도가 낮을 수 있다. 이 경우에 fill-in demon은 적중률이 떨어지는 priority table을 사용해서 prefetching하게 될 때, 이러한 prefetching 요구가 다시 priority table에 반영된다면 priority table은 적중률을 향상시키는 방향으로 업데이트되기 어려워진다. 따라서 fill-in demon의 prefetching요구는 priority table에 반영되지 않도록 해야 한다.

또한 package consumer가 요구한 페이지가 로컬디스크에 존재하는 경우에 NSS Configurator을 통해서 페이지에 대한 접근 요구정보를 정기적으로 서버에 보내주면 priority table의 정보는 더욱 정확해 질 것이다.

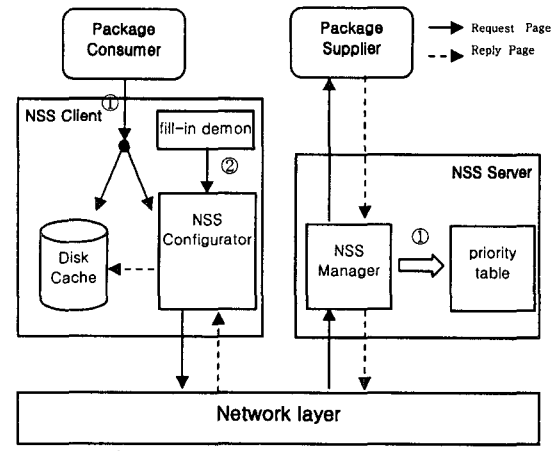

그림 6 client request에 대한 처리 흐름도

[그림 6]은 이에 대한 설명이다. 클라이언트의 실제 페이지 요구 ①은 페이지가 로컬 디스크 캐쉬에 있는 경우와 로컬 디스크 캐쉬에 존재하지 않는 경우로 나뉘어 진다. ①의 경우는 전자 혹은 후자를 막론하고 서버측의 priority table은 갱신되어야 한다. 하지만 ②와 같이 fill-in demon에 의한 prefetching을 위한 클라이언트 요청이라면 서버는 단지 해당 페이지를 전송 할 뿐 priority table을 수정하지는 않는다.

3. 메모리 관리

앞절에서 제안한 fill-in demon과 prefetching algorithm을 사용한다면 클라이언트 측에서는 속도의 향상을 가질 수 있지만 priority table을 모두 서버의 메모리에 적재 시켜야 해야 하는 문제가 생긴다. Priority table에서 하나의 엔트리는 inode number (4byte), page number(4byte), priority(2byte)로 이루어진다. 전체 크기가 400MB인 패키지를 위한 prioriry table의 경우 페이지의 크기가 4KB라면 관련 정보 갱신을 위해 100K개의 엔트리가 필요하므로, priority table의 메모리 용량은 1MB가 될 것이다. 서비스되는 패키지 수가 늘어날수록 서버는 그에 비례하여 더 많은 용량의 메모리를 부담하게 된다. 하지만 패키지를 실행할 때 필요한 page는 전체패키지 중 아주 일부분이므로, 클라이언트가 요구한 페이지에 대한 우선순위 정보만을 유지한다면 많은 메모리를 절약할 수 있을 것이다. 많은 시간이 지나면 동적으로 메모리를 할당한다고 해도 언젠가는 패키지 대부분의 페이지가 한번 이상 참조되어질 수 있을 것이다.

4. 결론 및 향후 연구

NSS 시스템에서 단순히 클라이언트가 요구하는 페이지만을 서버의 로컬디스크에서 읽어 클라이언트에게 전달한다면 네트워크 자원이 사용되지 않는 많은 시간이 낭비 된다. 본 논문에서는 네트워크 자원을 효율적으로 사용하고자 네트워크 자원이 사용되고 있지 않은 시간에 클라이언트가 다음에 사용할 가능성이 높은 페이지를 미리 서버에서 읽어와 클라이언트의 로컬디스크에 저장하는 background prefetching 기능을 설계하였다. 패키지 streaming에 적합한 효율적인 prefetching algorithm을 고려한 결과 본 논문에서는 dynamic page ordering 방법을 제안하였는데, 현재 진행중인 NSS의 구현이 끝나면 dynamic page ordering 방법의 성능평가에 들어갈 예정이다.

참 고 문 헌

[1] "리눅스 커널분석 2.4", 박장수 저, 가메출판사, 2003년.
[2] Z!stream White Paper, "http://www.softonnet.com/.
[3] An Enhanced Disk-Cashing NFS Implementation for Linux "http://www.cs.washington.edu/homes/gjb/doc/enhanced-linux-nfs-client/index.html".