가상 소프트웨어 실행 환경을 제공하는 주문형 소프트웨어 스트리밍 시스템 설계 및 성능평가

김 영 만[†] · 박 홍 재[††] · 한 왕 원[††] · 최 완[†††] · 허 성 진[††††]

요 약

소프트웨어 스트리밍은 프로그램 설치 및 실행을 위하여 서버로부터 프로그램 전송이 진행중인 동안에도 컴퓨터 상에서 미설치 소프트웨어의 실행이 즉각적으로 이루어지도록 하는 기능이다. 본 논문에서는 네트워크를 통하여 컴포넌트들의 자동 설치 기능들을 제공하며 프로그램 및 데이터 파일을 스트리밍하고 실행해주는 Software On-Demand(SOD)스트리밍 시스템을 제안한다.

제안된 시스템의 효용성을 입증하기 위하여 리눅스 상에서 즉각적인 소프트웨어 실행 환경과 함께 사용자가 소프트웨어 다운로드와 인스톨 작업에서 완전하게 벗어날 수 있도록 하는 SOD 시스템을 설계 및 구현하였다. 구현된 SOD 시스템은 복잡하고 실패하기 쉬운 설치 작업으로부터 사용자의 수고를 경감시키며 사용자가 UI 윈도우 또는 웹 브라우저를 통하여 look-and-click의 대화식 조작에 의해 여러 제품들을 쉽게 사용할 수 있도록 해주기 때문에 소프트웨어 개발자는 SOD 시스템에 기반한 가상 실행환경을 통하여 소프트웨어 제품을 광고하고, 전파할 새롭고 강력한 수단을 지원받게 된다. 또한 본 논문에서는 리눅스 상에서 두 가지 SOD 스트리밍 실험 환경을 구축한 후 성능평가 실험 결과에 대한 분석을 통해 향후에 SOD 시스템에 적용할수 있는 두 가지의 성능 개선 방법AIA(Application Initation Accelerator), SPP(Statistical Predictor Prefetching)를 제안한다.

키워드 : 주문형 소프트웨어 스트리밍, 네트워크 스트리밍 프로토콜, 성능실험 및 평가

Design and Performance Evaluation of Software On-Demand Streaming System Providing Virtual Software Execution Environment

Young Man Kim[†] · Hong Jae Park[††] · Wang Won Han[††] · Wan Choi[†††] · Seong Jin Heo[††††]

ABSTRACT

Software streaming allows the execution of stream-enabled software on desktop or portable computing devices like PC, PDA, laptop, cellular phone, etc., even while the transmission/streaming from the server may still be in progress.

In this paper, we present an efficient streaming system called Software On-Demand(SOD) streaming system to transmit stream-enabled applications in addition to automatic installation of program registry, environment variables, configuration files, and related components. In particular, we design and implement a SOD system in Linux to provide the user with the instant look-and-click software execution environment such that software download and installation are internally proceeded in a completely user-transparent way. Therefore, the SOD system relieves the user from the tricky, failure-prone installation business. In addition, the software developer now obtains a new, powerful means to advertise and propagate their software products since the user can use software packages via user-friendly UI window or web browser by look-and-click interactive operation.

In the paper, we also make a couple of SOD streaming experiments using a spectrum of popular softwares. Based on the analysis of the experiment results, we also propose two performance improvement schemes.

Key Words : Software On-Demand(SOD) Streaming, Network Streaming Protocol, Performance Experiment and Evaluation

1. 서 론

PC, PDA, 휴대용 개인 컴퓨터, 휴대 전화 등과 같이 데스크 탑 또는 이동식의 연산 장치에서 서버로부터 응용프로그램을 다운로드하고 설치하는 작업은 많은 시간이 소모되며, 윈도우나 리눅스와 같은 대표적인 컴퓨팅 환경에서의 공개 혹은 상업용 응용프로그램 설치 작업은 초보 사용자가 이해하기 힘든 지식을 요구한다. 그러나 이러한 문제는 서버로부터 응용프로그램의 전송 및 설치가 자동적으로 진행되도록 해주며 전체 소프트웨어의 다운로드 완료 전에도 해당 응용 프로그램이 실행될 수 있도록 해주는 소프트웨어

† 정 회 원 : 국민대학교 컴퓨터학부 교수
†† 준 회 원 : 국민대학교 일반대학원 전산과학 석사과정
††† 정 회 원 : ETRI 온디맨드 서비스 연구팀 팀장
†††† 정 회 원 : 창신대학 인터넷공학과 전임교수
논문접수 : 2005년 12월 5일, 심사완료 : 2006년 5월 30일

스트리밍 기술[8, 9]을 통해 해결 가능하다. 소프트웨어 스트리밍은 응용프로그램 실행에 불필요한 대부분의 블록들을 서버로부터 다운받지 않아도 프로그램 실행을 가능하게 해주기 때문에 메모리와 대역폭 같은 리소스를 매우 효율적으로 활용할 수 있게 된다. 즉, 본 논문에서 제안하는 스트리밍 방식을 사용하면 어떠한 데스크탑 PC 또는 이동식의 장치에서도 수많은 응용프로그램들을 별도의 설치과정 없이도 즉각 실행할 수 있다.

본 논문에서는 네트워크를 통하여 프로그램 등록, 환경변수 설정, 그리고 구성 파일과 관련된 컴포넌트들의 설치를 자동적으로 진행하며 소프트웨어를 스트리밍하고 실행해주는 Software On-Demand(SOD) 스트리밍 시스템을 제안한다. 제안된 시스템의 실현성을 입증하기 위해 리눅스 상에서 사용자가 소프트웨어 다운로드와 인스톨에서 완전하게 벗어날 수 있도록 하는 SOD 시스템을 설계 및 구현한다.

본 논문에서 제안된 SOD 시스템은 다음과 같은 특징을 가진다. 첫째, UI 윈도우에서 사용자가 실행하고자 하는 소프트웨어의 아이콘을 클릭하는 것만으로 소프트웨어 다운로드와 인스톨이 자동적으로 진행되기 때문에 SOD 시스템은 사용자가 복잡한 인스톨 작업으로부터 벗어나 본래의 작업에 집중할 수 있도록 해준다. 둘째, 사용자가 UI 윈도우 또는 웹 브라우저를 경유하여 룩-앤-클릭의 대화식 조작에 의해 새로운 소프트웨어 제품을 직관적으로 사용할 수 있도록 해주기 때문에 소프트웨어 개발자에게 새로운 제품의 광고와 유통을 위한 강력한 수단을 제공한다. 셋째, 설치 및 실행에 필요한 최소한의 프로그램 정보가 도착하면, 프로그램은 전체 패키지의 다운로드와 설치작업이 완료되지 않아도 실행을 시작하기 때문에 사용자는 새로운 프로그램을 실행하는 경우에 오랜 대기시간 없이 프로그램의 즉각적인 실행을 경험하게 된다. 넷째, SOD 스트리밍은 프로그램이 사용하고자 하는 데이터 파일을 모두 다운로드하지 않고도 필요한 페이지만 있으면 응용프로그램이 실행되도록 해준다. 마지막으로 SOD 스트리밍은 서버에서 직접 스트리밍되기 때문에 소프트웨어 배포와 업데이트를 용이하게 한다. 즉, 소프트웨어 개발자가 서버에 패치 정보를 저장하면 클라이언트 장치는 항상 소프트웨어의 최신 버전을 스트리밍 받게 된다.

본 논문에서는 SOD 스트리밍 시스템을 지원하는 스트림 서버로서 Z!Stream[16]서버를 사용한다. Z!Stream 서버는 마운팅, 인증, 감사작업, 접속 관리, 세션 관리, 부하 조절 등과 같은 스트리밍과 관련된 풍부한 함수 집합을 제공한다.

본 논문은 다음과 같이 구성된다. 2장에서는 소프트웨어 스트리밍에 관련된 연구들에 대하여 소개한다. 3장에서는 SOD 스트리밍 시스템을 제안한다. 4장에서는 리눅스에서 구현된 SOD 시스템에 대하여 행한 성능평가 실험 결과에 대하여 다룬다. 또한 결과 분석으로부터 SOD 시스템에 대한 두 가지 성능 개선 방법을 도출한다. 마지막으로, 5장에서는 SOD 시스템에 대한 향후 연구 과제를 소개하고 본 논문의 결론을 내린다.

2. 관련연구

소프트웨어 스트리밍과 유사한 관련기술로 다운로드, 원격 실행, 가상 파일시스템 및 소프트웨어 이동 실행 방식이 있다. 다운로드 방식은 네트워크로 연결된 컴퓨팅 환경에서 클라이언트가 서버로부터 임의의 소프트웨어를 요청, 다운로딩, 압축복원, 설치한 후에 프로그램을 실행하는 것이다. 이때 큰 프로그램인 경우 다운로드 시간이 매우 길어진다. 즉, 소프트웨어 다운로드(전송)시간이 응용프로그램 실행준비 시간의 대부분을 차지하게 되어 사용자는 오랜 대기 시간을 지루하게 보내야 한다.

다운로드 방식의 전형적인 예로는 파일 전송 프로토콜(FTP)[12]이 있는데 FTP 클라이언트는 서버로부터 전체 파일을 다운로드하게 된다.

다운로드와 유사한 방법으로 자바의 FileInputStream[10] 방식이 있다. 응용프로그램이 FileInputStream을 사용하여 파일을 읽을 때 자바 가상 머신(JVM)은 서버에서 전체 파일을 다운로드한다. 이때 큰 파일의 경우 다운로드 시간은 길어지며 다운로드가 끝나게 될 때까지 일시적으로 응용프로그램은 중지하게 된다.

원격 실행[1, 11]방식은 응용프로그램에 대한 원격 접속을 사용자에게 제공하는 서비스이다. 원격 실행에서 응용프로그램은 원격 장치에 저장되어 있으며 사용자 장치는 응용프로그램을 저장하거나 캐시하지 않는 상태에서 응용프로그램을 실행하기 위해 원격 장치에게 명령을 내리고 원격 장치는 클라이언트에게 실행결과를 보여주는 디스플레이 출력을 보낸다. 이러한 방법에서는 사용자의 입력 내용을 네트워크를 통하여 원격 응용프로그램에게 전달하는 시간이 추가되기 때문에 사용자 장치에서의 실행에 비해서 많은 시간이 걸린다. 원격 실행방식의 또 다른 문제는 서버의 제한된 자원 때문에 클라이언트가 증가하게 되면 각 클라이언트에 할당되는 시간이 줄어들어 효율적인 지원이 어려워진다는 것이다.

가상 파일시스템(VFS)방식은 클라이언트가 서버에 저장되어 있는 디렉토리를 마운트하여 액세스할 수 있도록 한다. 클라이언트는 마운트된 디렉토리안의 서버 파일에 자유롭게 접근할 수 있다. 원격 파일에 접근하기 위한 전형적인 과정을 설명해보면 우선, 사용자 프로세스가 시스템 호출 명령을 내리면 커널은 VFS에게 명령을 전달하고 VFS는 해당 요청을 처리한다. (예를 들어, 만약 read() 명령이 내려지면 VFS는 서버 또는 로컬 캐시로부터 데이터를 획득한 후에 사용자의 메모리에 적재한다.) VFS 중에서 가장 널리 사용하는 네트워크 파일시스템 (NFS)[5,14,15]는 네트워크 상에서의 파일 공유를 위해 클라이언트/서버 모델을 사용하고 파일을 처리하기 위해 원격 함수 호출 방법을 사용한다. 이와 같은 NFS 프로토콜을 사용하면 클라이언트는 서버에 위치한 파일 시스템에 액세스할 수 있다. 클라이언트는 파일

[2]을 캐시하기 위해 메인 메모리를 사용하므로 많은 네트워크 트래픽이 생성된다. 그 결과 NFS는 소수의 클라이언트만을 지원할 수 있으며 LAN 환경에서 잘 작동한다.

소프트웨어 이동 실행 기술은 프로그램의 전체 다운로드 없이도 클라이언트에서 응용프로그램이 동작할 수 있도록 해준다. 응용프로그램이 실행되는 동안 프로그램 코드는 클라이언트가 요구하는 경우 또는 백그라운드에서 전송된다.

Huneycutt et al.[6]는 PDA와 핸드폰과 같은 내장형 장치에서 제한된 네트워크 자원을 효율적으로 관리하는 방법을 제시하는데, 이는 소프트웨어 캐싱을 통하여 이루어진다. 이 시스템에서는 장치가 임의의 코드 페이지를 필요로 할 때, 해당 페이지가 캐시에 없는 경우 소프트웨어 캐시 실패가 발생하며 장치는 서버에게 해당 코드의 송신을 요청한다. 따라서 이 방법에서는 소프트웨어 캐시 실패가 끊임없이 발생하는 경우 프로그램은 빈번한 중단을 맞이하는 단점이 있다.

Raz, Volk 및 Melamed[13]는 응용프로그램을 여러 모듈들의 집합으로 나누어 긴 부하시간으로 인한 지연문제를 해결하는 방법을 기술하였다. 예를 들면, 하나의 자바 애플릿은 여러 자바 클래스들로 이루어진다. 최초의 모듈이 로드되어 응용프로그램이 모듈을 실행하는 동안 추가 모듈이 백그라운드로 스트리밍되어 차후의 실행이 곧바로 이어지도록 하고 있다. 전송시간의 감소를 위하여 streaming stub 프로시져를 최적화하여 사용한다. 이때 소프트웨어 전달 단위가 다양한 크기를 가진 모듈이므로 코드 전송에 따른 정지 시간을 예측하는 것은 어렵다.

Krintz et al.[7]는 모바일 프로그램 실행을 위한 전송 방법을 제안한다. 이 모델에서는 자바 플랫폼을 실행 환경으로서 사용하며 클래스 파일을 전역 데이터와 지역 변수가 포함된 함수코드로 나누어 전역 데이터를 우선 전송하고 그 다음 실행 클래스 파일을 구성하는 각 함수를 송신한다. 자바 프로그램 실행 시 임의의 함수가 호출될 때 그 함수 또는 관련 데이터의 전송이 완료되지 않았다면 프로그램은 함수 코드 또는 데이터가 전송되기 전까지 대기하게 된다.

Hartman et al.[4]는 네트워크에서 동적으로 움직이는 함수에 대한 아이디어를 소개한다. 이 방식에서는 여러 종류의 플랫폼에서 실행될 수 있는 위치 독립적인 이동식 코드를 사용한다. 이동식 코드는 장치간을 이동하면서 실행된다. 이동식 코드가 새로운 장치에서 신속히 처리되기 위해서는 모든 장치상에서 이동식 코드를 내부 실행 코드로 번역하는데 있어 빠른 컴파일러가 필요하다.

소프트웨어 스트리밍은 소스코드 레벨에서도 제공될 수 있다. 예를 들면, 소스 코드가 내장형 장치에 전송되어 로드시간[3]에 컴파일되는 방식이 있다. 비록 소스 코드가 컴파일된 바이너리 이미지보다 작고 전송 시간이 빠르더라도, 도착된 소스를 컴파일 하는데 걸리는 시간은 매우 길어질 수 있으며 컴파일러가 사용하는 임시 파일들을 저장하기 위한 메모리 공간의 크기도 필요하게 된다. 더욱이 컴파일러는 클라이언트 장치에서 메모리에 상주해야만 하므로 큰 용량의 기억공간을 차지한다. 따라서 이 방법은 작은 메모리를 가지거나 느리고 낮은 파워의 내장형 프로세서와는 어울리지 않는다.

Embedded Software Block Streaming (ESBS)[8, 9]방법에서는 내장형 장치에서 효율적으로 스트리밍될 수 있도록 내장형 응용프로그램 자체가 수정되어야 한다. 스트리밍이 가능한 내장형 소프트웨어는 미리 컴파일 되어 완성된 바이너리 이미지로부터 생성된다. 스트리밍 가능한 처리를 위해 내장형 소프트웨어에 대한 지식이 필요하기 때문에 전문지식을 가진 사람만이 이 작업을 수행할 수 밖에 없다. 응용프로그램의 첫 번째 블록이 다운로드되면 Softstream Loader는 할당된 메모리에 블록을 로드하고, 블록 검색 테이블에 블록의 주소를 저장한다. Softstream Loader가 종료된 후 스트리밍된 블록내의 응용프로그램 코드가 실행된다. 만일 실행에 필요한 블록이 메모리에 없다면 스트리밍 요청을 하고 블록이 메모리에 있으면 블록 검색 테이블에서 조사한다. 필요한 블록이 메모리에 로드 된 후 Softstream Loader는 실행 예정된 주소로 이동하여 프로그램의 실행이 재개된다. 이 방법에서 채용된 실행시간 중의 동적 코드 수정은 많은 파일과 컴포넌트로 이루어진 복잡한 소프트웨어 패키지로 구성된 상업용 소프트웨어 도구에서는 긴 시간의 실행 지연을 초래한다. 더욱이 ESBS는 단일 쓰레드형 소프트웨어를 위해 설계되어 있어 멀티 쓰레드형 응용프로그램에서는 사용하지 못한다. 환경 변수, 구성 파일과 소프트웨어 레지스트리는 네트워크 스트리밍을 통한 전반적인 소프트웨어 실행에 있어서 해결해야 할 또 하나의 과제이다. 그러나 ESBS는 이 문제에 대해 어떠한 해결책도 제공하지 않고 있다.

3. Software On-Demand(SOD) 스트리밍 시스템

이상적인 응용프로그램 컴퓨팅 환경에서는 사용자가 다운로드, 인스톨 등과 같은 부가적인 업무에 시간과 노력을 투입하지 않아도 본 업무를 처리할 수 있도록 부수적인 작업을 자동으로 처리하는 기능의 지원이 바람직하다. 사용자가 주어진 과제를 해결하기 위해 특정 응용프로그램을 실행하고자 할 때 데스크 탑 상에서 해당 응용프로그램을 상징하는 아이콘을 클릭하면 프로그램 초기 실행에 필요한 모듈들이 메모리에 로딩 되자마자 프로그램의 UI 윈도우가 화면에 나타나게 되며 이 프로그램이 해당 환경에서 최초로 호출되는 경우에는 추가적으로 프로그램 다운로드와 인스톨 절차가 백그라운드에서 진행되어야 한다.

이번 장에서는 위에서 열거한 기능을 제공하는 Software On-Demand(SOD) 스트리밍 시스템을 제안한다. SOD 스트리밍 시스템은 소프트웨어 인스톨 절차의 자동실행에 따른 사용자 편리성을 향상시키는 구조로 되어 있는데 다음 절에

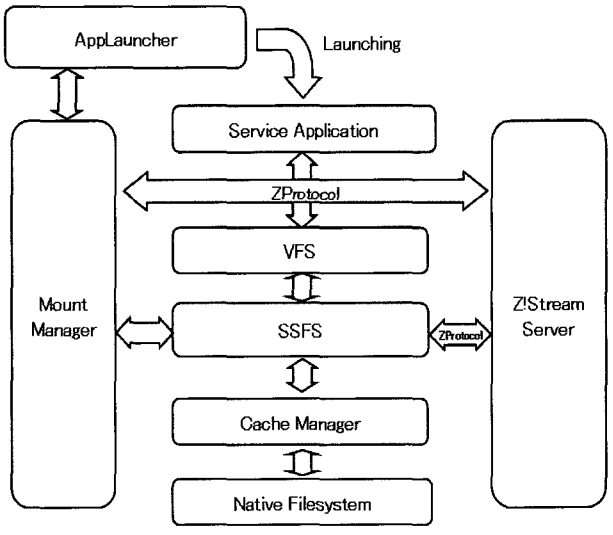
서 자세히 설명한다.

3.1 SOD 스트리밍 시스템 구조

(그림 1)은 SOD 스트리밍 시스템의 구조를 보여주고 있는데 가상 컴퓨팅 환경을 제공하여 다운로드와 인스톨 절차를 사용자로부터 은폐시킨다. 가상 컴퓨팅 환경의 주된 목적은 각 응용프로그램 소프트웨어를 위해 프로그램에 종속적인 레지스트리, 환경 변수, 구성 파일, 관련 컴포넌트에 대한 컴퓨팅 환경을 사용자가 인식하지 않는 상태에서 자동적으로 설정 · 구축하는데 있다.

사용자가 새로운 응용프로그램을 최초로 선택하면 SOD 시스템은 환경 관련 데이터를 스트리밍 서버로부터 수신하여 처리한다. 환경 설정이 끝나게 되면 SOD 시스템은 서버에 프로그램 실행을 위한 최소한의 바이너리 실행 페이지들을 요청한다.

SOD 사용자 인터페이스 프로그램인 응용프로그램 실행기(AppLauncher)은 SOD 환경에서 실행가능한 소프트웨어 아이콘들을 보여준다. 사용자가 응용프로그램 실행기 윈도우에서 임의의 아이콘을 선택하면 SOD 시스템 컴포넌트 중의 하나인 마운트 매니저(MM)는 응용프로그램 환경 데이터를 다운로드하고 인스톨하는 작업을 수행한다. MM은 이러한 작업을 위해 Z 프로토콜을 이용하여 해당 데이터를 스트림 서버에 요청한다. 응용프로그램 컴퓨팅 환경에서 응용프로그램을 위한 초기 설정이 끝나면 응용프로그램 실행기는 응용 프로세스를 시작하기 위하여 실행 이미지 블록들을 운영체제에게 요청한다. 이때 프로그램 이미지와 데이터 파일을 위한 I/O가 발생하고 해당 요청은 리눅스 가상 파일시스템(VFS) 모듈을 경유하여 SOD 시스템의 세번째 컴포넌트인 스트림 서비스 파일시스템(SSFS)에 도착한다. SSFS는 우선 SOD 시스템의 네번째 컴포넌트인 캐시 매니저(CM)를 경유하여 로컬 디스크 캐시를 검색한다. 만일 페이지가 로컬 디스크 캐시에서 발견되지 않으면 SSFS는 스트리밍 서버에게 해당 페이지 요구 메시지를 보내기 위해 Z 프로토콜을 통해 서버로부터 요구한 페이지를 전송받는다. SSFS를 통하여 입수된 페이지는 재사용을 위해 로컬 디스크 캐시에 저장되고 동일한 내용이 메모리 캐시에도 생성된 후 프로그램 실행이 중지된 위치에서 재개된다. 현재 SOD 시스템은 리눅스 상에서 UI 기능을 제외한 핵심 모듈들을 구현 완료된 상태에 있다.



(그림 1) SOD 스트리밍 시스템 구조

3.2 SOD 스트리밍 시스템 모듈

이 절에서는 본 시스템을 구성하는 SOD 각 모듈에 대하여 자세히 설명한다.

3.2.1 응용프로그램 실행기(AppLauncher)

응용프로그램 실행기는 사용자가 원하는 응용프로그램을 손쉽게 찾아내고 선택할 수 있도록 웹 브라우저와 유사한 유저 인터페이스를 제공한다. 사용자가 실행하고자 하는 응용프로그램을 상징하는 아이콘을 클릭할 때, 응용프로그램 실행기는 필요시 마운트 매니저에게 응용프로그램 실행을 위한 가상 컴퓨팅 환경을 새로 구축하도록 지시한다. 그 후 fork()와 exec() 시스템 호출에 의해 응용프로그램 프로세스를 개시한다.

3.2.2 마운트 매니저(MM)

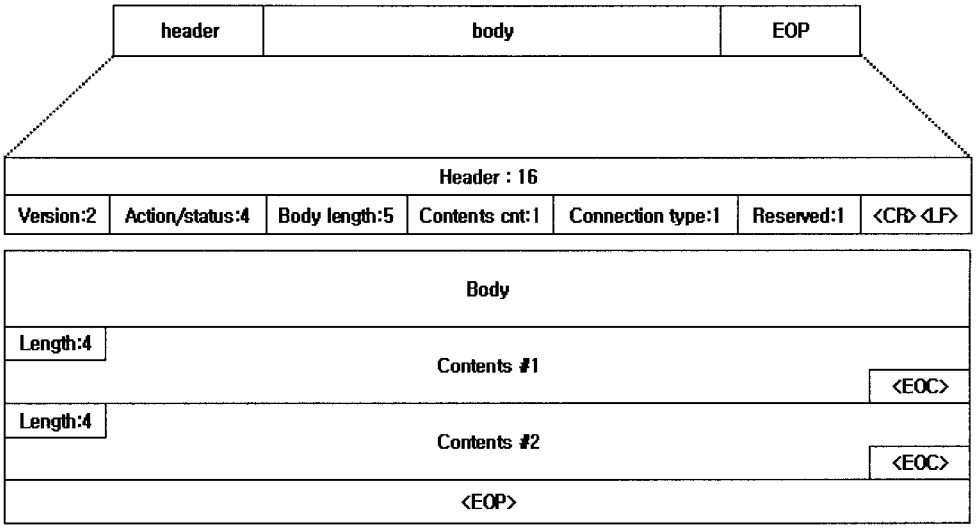
리눅스는 다중 사용자용 실행 환경을 제공한다. 그러므로 MM 역시 다중 응용프로그램 실행기 프로세스 정보를 유지해야 한다. 각 응용프로그램 실행기가 개별 응용프로그램 컴퓨팅 환경을 생성하도록 MM에게 지시하면 MM은 로컬 캐시로부터 해당 프로그램용 환경 데이터를 검색한다. 만일 데이터가 로컬 캐시에서 발견되지 않으면, MM은 스트리밍 서버에게 환경 데이터 요구 메시지를 보낸다. MM은 도착된 환경 데이터를 사용하여 가상 컴퓨팅 환경을 구축하고 관리한다.

3.2.3 스트리밍 서비스 파일시스템(SSFS)

SSFS는 리눅스 VFS와 네이티브 파일 시스템(예를 들면, ext2) 사이에 위치하는 SOD전용 가상 파일 시스템 모듈이다. 응용프로그램 프로세스가 시작하여 데이터 페이지를 요구할 때 해당 호출은 리눅스 VFS를 경유하여 SSFS에 도착한다. SSFS는 우선 캐시 매니저(CM)를 호출하여 로컬 디스크 캐시내에서 해당 페이지를 찾는다. CM은 페이지의 내용 혹은 미발견 오류정보를 SSFS에게 리턴한다. 후자의 경우 SSFS는 해당 페이지 데이터를 얻기 위해 스트리밍 서버에게 요구하며 서버로부터 새로운 페이지 데이터가 도착하면, 해당 페이지는 CM을 경유하여 로컬 디스크 캐시에 저장되고, 메모리에도 로딩된 후에 응용프로그램 프로세스 혹은 MM의 실행이 재개된다.

3.2.4 캐시 매니저(CM)

데이터 재사용성을 증진시키기 위해 로컬 디스크 캐시에



(그림 2) Z 프로토콜 포맷

저장되는 데이터는 응용프로그램 스트리밍 서비스가 보다 신속히 진행될 수 있도록 도와준다. 즉, 응용프로그램 프로세스는 네트워크 페이지 요구-응답 지연시간을 생략함으로써 사용자에게 신속한 프로그램 실행을 제공하며 귀중한 네트워크 자원을 절약한다. CM은 빠른 페이지 검색을 위해 내부적으로 인덱스 구조를 가지고 있다.

3.3 Z Protocol

Z 프로토콜은 SOD 스트리밍을 위해 설계된 응용 프로토콜이다. 실시간 QoS 필요조건을 만족시켜야만 하는 멀티미디어 스트리밍과는 대조적으로 SOD 스트리밍은 신뢰할 수 있는 메시지 전달을 필요로 한다. 따라서 Z 프로토콜은 TCP 프로토콜 상위에 구축된다.

메시지에는 제어와 데이터 메시지 2가지 종류가 있다. 제어(데이터) 메시지는 RASCP(RASP) 보조 프로토콜에서 정의되고, MM(SSFS)과 스트리밍 서버 사이에서 사용된다. MM은 RASCP를 경유하여 환경 데이터, 인증, 접속 관리, 세션 관리 등의 제어 정보를 송신한다. 반면에 SSFS는 프로그램 또는 데이터 페이지 요구 메시지를 RASP을 사용하여 보낸다.

RASCP와 RASP 모두 공통의 요구-응답 통신 패러다임을 따르므로 공통 Z 프로토콜 헤더 포맷을 공유한다. Z 프로토콜의 포맷은 (그림 2)와 같이 구성된다.

4. SOD 스트리밍 시스템 성능평가

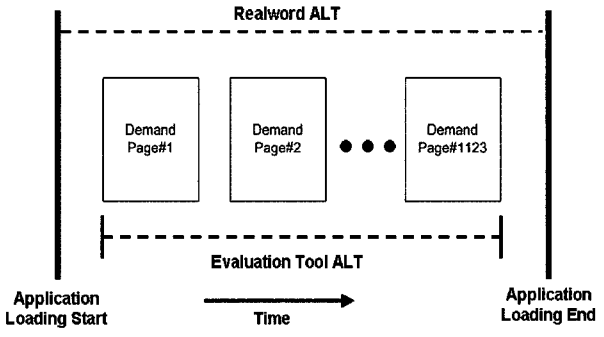
이 장에서는 3장에서 설계 및 구현한 SOD 스트리밍 시스템에 대하여 성능 평가한다. SOD 스트리밍 시스템 사용자의 만족도를 측정하는 성능평가 실험을 행하고 그 결과분석을 통해 사용자에게 가상 컴퓨팅 환경에서 보다 높은 서비스 만족도를 제공할 수 있는 성능개선 방법에 대하여 제안한다.

4.1 성능평가 지수

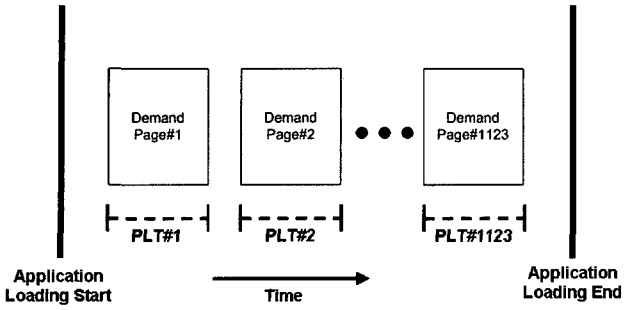
대화형 응용프로그램 환경에서 신속한 응용프로그램 실행 속도를 나타내는 성능평가 지수로서 사용자 인터페이스 화면에 관련된 Application Load Time(ALT), Page Load Time(PLT) 그리고 Initial Execution Loading Size(IELS)를 선정하였다.

ALT는 임의의 프로그램이 마우스에 의하여 실행 선택된 직후부터 초기 사용자 인터페이스 윈도우가 화면에 나타날 때까지의 경과시간이다. ALT는 환경 데이터의 크기, 프로그램의 초기화 또는 런칭에 관련된 데이터 크기에 직접적으로 비례하고 전송 매체의 속도에 반비례한다. 따라서 낮은 ALT 값은 프로그램이 빨리 실행된다는 것을 뜻한다.

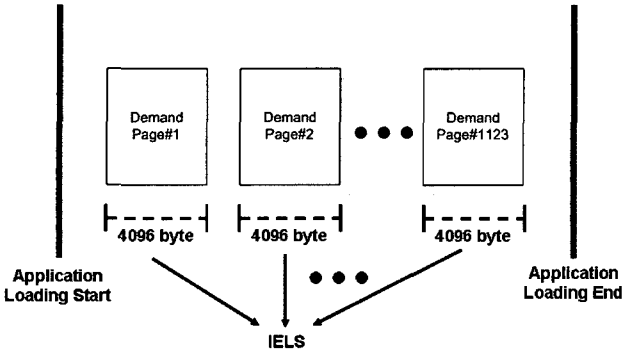
실세계에서의 ALT는 프로그램이 선택된 직후부터 사용자 윈도우가 화면에 나타날 때까지의 절대적인 시간이지만, 성능평가 도구에서 계산되어지는 ALT는 실세계의 ALT와는 약간의 차이가 있다. (그림 3)에서 보이는 바와 같이 성



(그림 3) 성능평가 도구 ALT



(그림 4) 성능평가 도구 PLT



(그림 5) 성능평가 도구 IELS

능평가 도구의 ALT는 첫 번째 페이지가 요구된 시간부터 사용자 윈도우가 화면에 나타나기 직전에 요구된 페이지가 처리될 때까지의 시간을 나타내므로 실세계의 ALT 보다 약간 적은 시간이 된다.

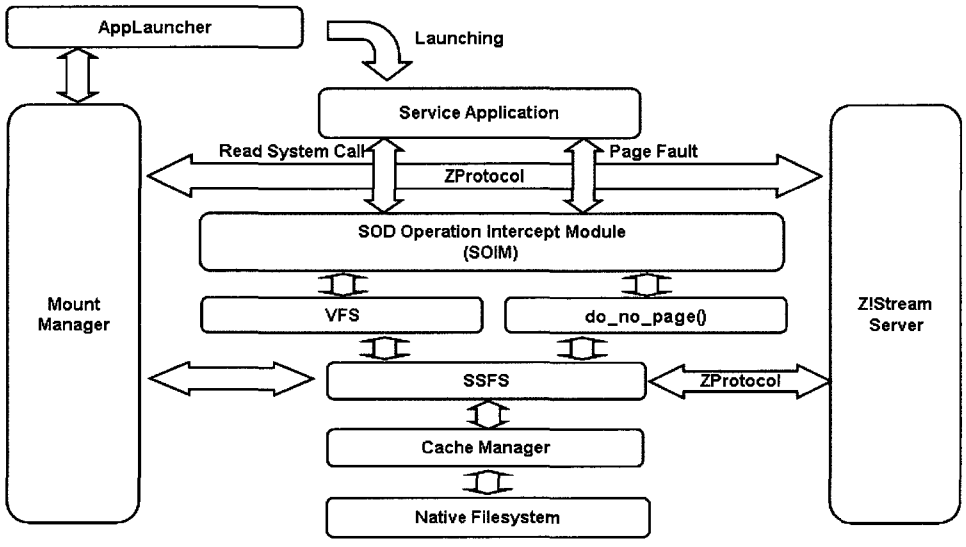
PLT는 임의의 프로그램에 의해서 요구된 페이지가 저장매체 혹은 네트워크로부터 읽혀지고 처리되는 시간을 나타낸다. ALT 동안에 요구된 페이지들에 대한 각각의 PLT가 로그파일에 기록되고, PLT의 평균을 계산함으로써 프로그램의 PLT가 결정되어진다. (그림 4)를 예로 들면 ALT동안 1123번의 페이지 요구가 있었고, 1123번의 각 페이지 요구에 대해 PLT가 기록되고 프로그램에 대한 PLT는 1123개 PLT의 평균을 계산하므로써 구할 수 있다. 낮은 PLT는 결과적으로 ALT를 작아지게 하므로 PLT가 낮을수록 프로그램은 빨리 실행된다는 것을 뜻한다.

마지막 성능평가 지수인 IELS는 ALT동안에 저장매체 혹은 네트워크로부터 읽혀진 총 바이트 수를 의미한다. (그림 5)에서 보는 바와 같이 페이지 요구시 읽혀지는 바이트 수를 모두 더해서 ALT동안의 IELS를 결정한다. IELS가 크다는 것은 프로그램이 실행되기 위해 읽혀져야 하는 바이트 수가 크다는 것을 의미하므로 IELS가 크면 클수록 프로그램은 늦게 실행될 것이다.

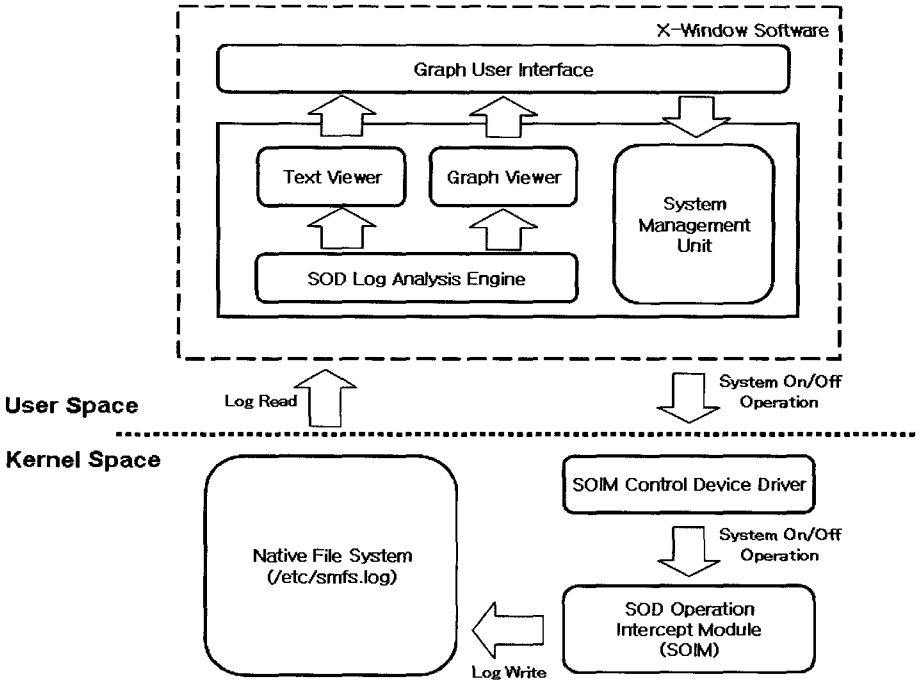
4.2 성능평가 모듈설계 및 구현

이번 절에서는 앞에서 설명한 SOD 시스템의 성능을 평가하는 성능평가용 소프트웨어를 설계하고 실제 구현이 어떻게 이루어졌는가를 설명한다.

SOD 성능평가 소프트웨어의 핵심부분인 SOD Operation Intercept Module(SOIM)은 SOD 응용프로그램들이 SSFS를 사용하기 위해 호출하는 시스템 콜을 가로채고 연산에 대한 로그를 기록하는 역할을 담당한다. SOIM은 파일시스템에 대한 읽기 연산을 수행하는 시스템 콜과 페이지 폴트 발생시 페이지 폴트를 처리하는 시스템 콜에 대한 로그를 기록함으로써, 각 성능평가지수를 측정할 수 있도록 도와준다. SOIM은 커널 모듈로 구현되어지는데 커널 모듈이란 필요에 따라 커널에 로드하거나 언로드 할 수 있는 특정한 기능을 수행하는 코드이다. 이렇게 함으로써 쉽게 SOIM을 커널의 기능으로써 확장할 수 있을 뿐만 아니라 운영체제를 다시 부팅 하지 않고도 로그 기능을 수행할 수 있도록 만들 수 있다. (그림 6)에 나타난 것과 같이 SOD와 관련된 시스템 콜들은 운영체제에 의하여 처리되기 전에 SOIM 모듈이 가로채어 진다. 시스템 콜을 가로챈 SOIM 모듈은 연산에 대



(그림 6) SOD 성능평가 모듈



(그림 7) SOD 성능평가 소프트웨어 구조

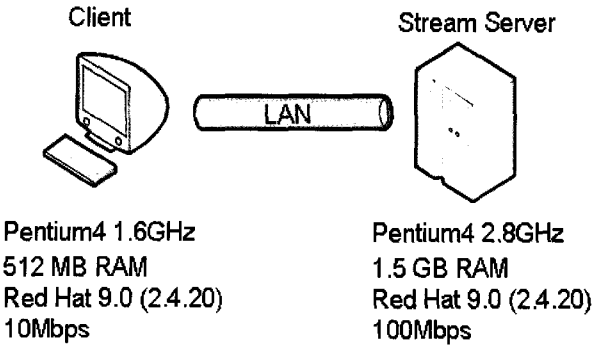
한 로그를 기록하고 SOD 연산이 제대로 실행될 수 있도록 원래 시스템 콜이 처리되는 부분으로 시스템 콜을 전달한다.

(그림 7)은 SOD 성능평가 소프트웨어의 전체 구조를 나타내고 있다. SOD 성능평가 소프트웨어는 크게 SOD 응용프로그램의 연산을 가로채어 파일에 로그를 기록하는 SOIM, 파일에 기록된 SOD 연산의 로그를 분석하기 위한 SOD Log 분석 엔진, 성능평가 소프트웨어의 On/Off와 같이 시스템과 관련된 기능들을 수행하고 관리하는 System Management Unit 그리고 성능평가 담당자에게 평가결과를 보여주는 GUI로 이루어진다.

SOD 성능평가 소프트웨어의 실행흐름은 사용자가 성능평가 소프트웨어의 GUI를 사용하여 로그기록의 시작을 요구하면 System Management Unit은 커널 디바이스 드라이버로 구현된 SOIM Control Device Driver를 통해 SOIM에게 로그를 기록하도록 시킨다. 로그기록을 지시받은 SOIM은 SOD 응용프로그램의 파일시스템 관련 연산이 요청되어질 때 마다 형식에 맞추어 로그파일에 로그를 기록한다. 기록되어진 로그는 SOD 로그분석 엔진에 의해 읽혀져서 분석된다. SOD 로그는 연산을 요청한 시간, 연산이 완료된 시간, 연산을 요구한 프로세스의 ID, 연산의 종류, 요구된 블록의 크기 그리고 연산의 수행방향을 기록한다. 연산의 수행방향이란 SOD 시스템에서 파일 페이지를 실제로 가져오는 위치를 말한다. SOD 시스템에서 파일을 읽어오는 위치로는 세군데가 있는데, 첫째 네트워크를 통해 SOD 서버로부터 원하는 파일의 일부분을 얻어 올 수 있는 것이다. 두번째는 클라이언트의 로컬 파일시스템에 존재하는 디스크 캐시에서 원하는 파일의 페이지를 읽어올 수 있다. 나머지 위치는 커널에서 제공하는 메모리 캐시에서 파일의 페이지를 읽어오는 것이다. 로그파일에 기록된 로그는 텍스트 뷰어나 그래픽 뷰어에 의해 사용자의 GUI에 보여지게 된다.

4.3 성능평가를 위한 실험환경

(그림 8)은 SOD 시스템의 성능평가를 위한 실험환경을 나타내고 있는 그림이다. 서버와 클라이언트는 랜을 통해서 연결되어 있고 하드웨어 사양은 그림에 나타난바 와 같으며, 본 논문에서는 좀 더 정확한 데이터를 얻기 위해 Abiword, Volley, Acrobat 세가지 프로그램의 ALT, PLT 그리고 IELS를 여러번 측정한 결과들의 평균값을 각 프로그램의 최종 값으로 선택하였다. SOD 성능평가 소프트웨어에 의해서 성능평가 되어지는 프로그램의 설치용량은 <표 1>과 같다.



(그림 8) SOD 성능평가 실험환경

〈표 1〉 성능평가되는 프로그램의 설치용량

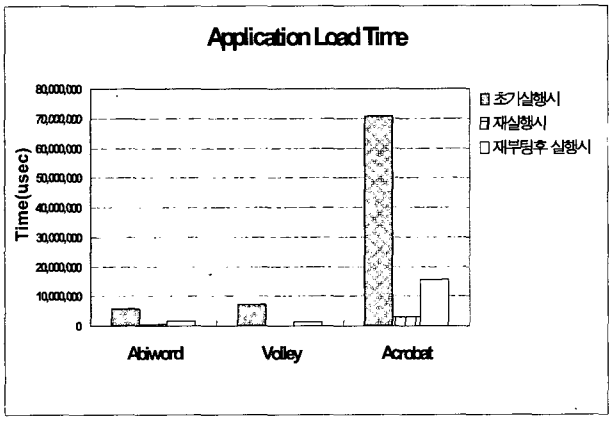
	Acrobat	Volley	Abiword
전체용량	151.78MB	7.37MB	30.12MB

4.4 실험결과 및 분석

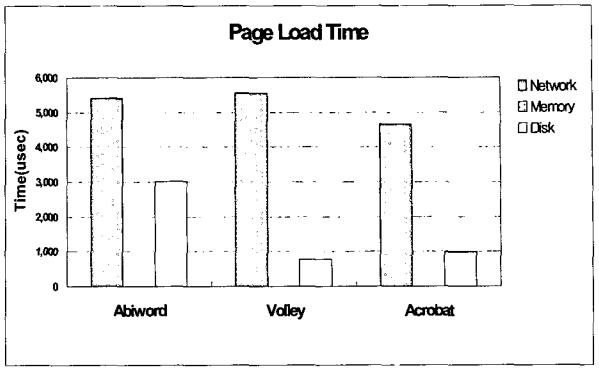
이번 절에서는 여러 상황 하에서 ALT를 측정한 결과를 소개한다. 측정 데이터의 정확성을 높이기 위해 측정한 데이터를 여러 차례 반복하여 이들의 평균을 구하였다. 우선 새로운 응용 소프트웨어의 최초 실행시의 초기 ALT를 측정하였다. 또한 동일한 프로그램에 대하여 재실행시 페이지 캐싱 메커니즘이 가진 성능향상 정도를 밝히기 위하여 재실행 ALT 값을 측정하였다. 또한 IELS, 즉 최초의 ALT 기간 동안 다운로드 되는 패킷 데이터의 크기 또한 측정하였다.

(그림 9)는 프로그램이 사용자에 의하여 실행 선택된 직후부터 초기 사용자 인터페이스 윈도우가 화면에 나타날 때까지의 경과시간인 ALT를 측정한 결과값을 나타내고 있다. ALT는 각 프로그램에 대하여 초기실행시, 재실행시, 재부팅 후 실행시의 세 경우에 대하여 측정하였다. 초기실행시에는 디스크 캐시나 메모리 캐시에 요구된 페이지가 존재하지 않으므로 페이지를 네트워크를 통해 얻어오게 된다. 그리고 초기실행한 이후에 바로 프로그램을 재실행하게 되면 한 번 얻어온 페이지는 이미 메모리 캐시에 저장되어 있으므로 페이지를 메모리 캐시에서 읽어오게 된다. 마지막으로 컴퓨터를 재부팅하고 프로그램을 실행하면 메모리 캐시가 재부팅시에 플러쉬 되었으므로 메모리 캐시에는 아무 페이지가 남아있지 않게된다. 하지만 이전에 네트워크를 통해 얻어온 페이지는 디스크 캐시에 저장되어 있으므로 해당 페이지를 네트워크가 아닌 로컬 디스크 캐시로 부터 읽어오게 된다. (그림 9)에서 보는 바와 같이 ALT는 재부팅후의 실행시가 초기실행시보다 낮고, 재부팅후 실행시 보다 재실행시가 낮은 것을 확인 할 수 있다. 즉 네트워크 보다는 디스크 캐시, 디스크 캐시 보다는 메모리 캐시에서 페이지를 얻어오는 것이 프로그램이 빨리 실행 된다는 것을 의미한다.

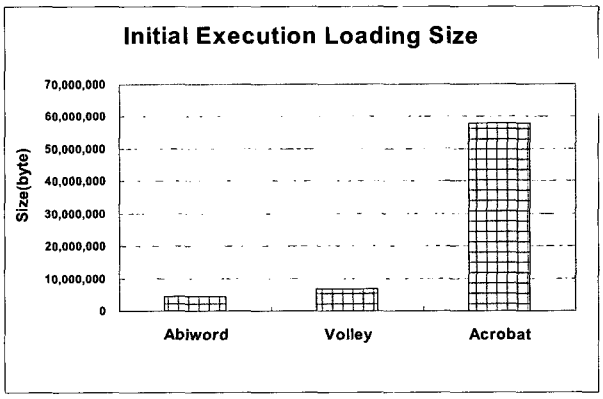
(그림 10)에서 보는 바와 같이 하나의 페이지를 읽어오는



(그림 9) Application Load Time 비교



(그림 10) Page Load Time 비교



(그림 11) Initial Execution Loading Size 비교

데 소요되는 시간을 나타내는 PLT 또한 ALT와 마찬가지로 네트워크 보다는 디스크가, 디스크 보다는 메모리가 더 적은 시간이 걸리는 것을 확인할 수 있었다.

마지막으로 IELS는 Abiword의 경우 4.5MB, Volley의 경우 6.8MB, Acrobat의 경우 57.8MB를 요구하는 것을 (그림 11)에서 확인 할 수 있다.

결론적으로 ALT는 환경 데이터의 크기, 프로그램의 초기화 또는 런칭에 관련된 데이터 크기를 의미하는 IELS와 직접적으로 비례하고, 하나의 페이지를 읽어오는데 소요되는 시간인 PLT와는 비례하는 것을 확인 할 수 있었으며, 전송매체의 속도와 ALT가 반비례 관계라는 것을 확인 할 수 있었다.

4.5 성능 개선 방법

본 절에서는 앞 절에서 설명한 실험 결과의 분석으로부터 차후에 SOD 스트리밍 시스템의 성능을 향상시키기 위해 적용할 수 있는 두 가지 성능 개선 방법을 제안한다.

임의의 프로그램이 마우스에 의하여 실행 선택된 직후부터 최초의 UI 윈도우가 화면에 나타날 때까지 다운로드 되어야 하는 페이지 전체를 백그라운드에서 최대 네트워크 속도로 다운로드한다면 ALT의 최소화가 이루어 질 수 있을 것이다. 이 제안방법을 Application Initiation Accelerator (AIA) 방법이라고 부른다. Statistical Predictor Prefetching (SPP)으로 불리는 또 다른 성능 개선 방법은 자주 액세스하

는 페이지를 접근 빈도와 시간 순서에 따라 스트리밍 서버에 기록한 후, 서버는 이 페이지를 백그라운드를 통해 SOD 시스템으로 송신하여 로컬 캐시에 저장하도록 한다. 이러한 페이지는 사용자 응용프로그램 프로세스가 조만간 접근할 가능성이 높은 부분이며, 앞 절의 PLT 측정을 통해 확인한 상대적으로 느린 네트워크가 아닌 로컬 캐시를 통해 페이지를 가져오기 때문에 보다 빠른 사용자 응용프로그램의 실행을 가져올 수 있을 것이다.

5. 결 론

본 논문에서는 프로그램 등록, 환경 변수 설정, 구성 파일 및 관련 컴포넌트의 자동 인스톨 기능을 자동적으로 실행함으로서 응용프로그램의 신속하고 편리한 스트리밍 실행 서비스를 제공하는 Software On-Demand(SOD) 스트리밍 시스템을 제안하였다.

제안된 SOD 시스템은 다음과 같은 특징을 가진다. 첫째로, SOD 시스템은 복잡한 인스톨 작업으로부터 사용자의 수고를 덜어준다. 두 번째로, 소프트웨어 개발자에게 새로운 제품의 광고와 유통을 위한 강력한 수단을 제공한다. 세 번째로 사용자는 새로운 프로그램을 실행하는 경우에 UI 윈도우의 빠른 팝업을 경험하게 된다. 네 번째로 SOD 스트리밍은 프로그램이 사용하는 데이터 파일을 모두 다운로드하지 않고도 응용프로그램이 데이터 파일을 액세스할 수 있도록 해준다. 마지막으로 SOD 스트리밍은 서버에서 직접 스트리밍되기 때문에 소프트웨어 배포와 업데이트를 용이하게 한다.

본 논문에서 제안된 SOD 시스템의 효용성을 보여주기 위하여 다양한 응용프로그램 소프트웨어에 대한 SOD 스트리밍 실험을 통하여 성능측정 작업을 수행하였으며, 측정 결과의 분석을 통해 두가지 성능 개선 방법을 제안하였다. Accelerator Initiation Accelerator(AIA) 방법은 응용프로그램의 초기 로드시간 ALT를 최소화하며, Statistical Predictor Prefetching(SPP) 방법은 응용프로그램의 대기 시간을 현저하게 줄일 것으로 판단되는데 향후 제안한 두가지 성능 개선 방법을 구현하고 그 성능을 입증하기 위한 실험을 실시할 것이다.

참고문헌

[1] Barron, D., *The World of Scripting Languages*, Chichester, NY, Wiley, 2000.

[2] Callaghan, B., *NFS Illustrated*, Reading, MA, Addison-Wesley, 2000.

[3] Eisenhauer, G., Bustamante, F., and Schwan, K., "A middleware toolkit for client-initiated service specialization," *Operating Systems Review*, Vol.35, No.2, pp.7~20, 2001.

[4] Hartman, J., Manber, U., Peterson, L., and Proebsting, T., "Liquid software: a new paradigm for networked systems," Tech. Rep. 96-11, *Department of Computer Science, University of Arizona*, Tucson, AZ, June 1996.

[5] Herman, D., *UNIX System V NFS Administration*, Englewood Cliffs, NJ, PTR Prentice Hall, 1993.

[6] Huneycutt, C., Fryman, J., and Mackenzie, K., "Software caching using dynamic binary rewriting for embedded devices," *Proceedings of International Conference on Parallel Processing*, pp.621~630, 2002.

[7] Krintz, C., Calder, B., Lee, H., and Zorn, B., "Overlapping execution with transfer using non-strict execution for mobile programs," *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.159-169, 1998.

[8] Kuacharoen, P., Mooney, V., and Madisetti, V., "Software streaming via block streaming," *Proceedings of the Design Automation and Test in Europe*, pp.912-917, Mar., 2003.

[9] Kuacharoen, P., Mooney, V., and Madisetti, V., "Efficient Execution of Large Applications on Portable and Wireless Clients," *Proceedings of the Mobility Conference & Exhibition*, Aug., 2004.

[10] Lindholm, T. and Yellin, F., "*The Java Virtual Machine Specification, Reading*," MA: Addison-Wesley, second ed., 1999.

[11] Pavlidis, T., *Fundamentals of X programming: graphical user interfaces and beyond*, New York, NY, Kluwer Academic, 1999.

[12] Postel, J. and Reynolds, J., "FILE TRANSFER PROTOCOL (FTP)," RFC 959. *The Internet Engineering Task Force*, Oct., 1985. http://www.ietf.org/rfc/rfc0959.txt?number=959.

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

[14] Santifaller, M., *TCP/IP and NFS: internetworking in a UNIX environment*, Reading, MA, Addison-Wesley, 1991.

[15] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D., "Network File System (NFS) version 4 Protocol," RFC 3530. *The Internet Engineering Task Force*, Apr., 2003. http://www.ietf.org/rfc/rfc3530.txt?number=3530.

[16] SOFTonNET Inc., "Z!Stream Technology," http://www.softonnet.com.

김 영 만

e-mail : ymkim@kookmin.ac.kr

1976년3월~1980년2월 서울대 공과대학 기계공학과

1980년3월~1982년8월 한국 과학기술원 기계공학과

1982년3월~1983년2월 중소기업진흥공단

1983년3월~1986년2월 LG전자 연구원(Robot, NC M/C, Automation, Compiler/OS, Fiber Network)

1992년11월~1996년8월 Mitsubishi Materials Corp., Central Research Lab. 초빙연구원(공장자동화를 위한 필드네트워크 설계/개발)

2001년9월~2004년2월 Ohio State Univ. 초빙 연구원 (DARPA NEST 센서네트워크 미들웨어 프로젝트 수행)

1996년3월~현재 현재 국민대학교 컴퓨터학부 교수

관심분야 : USN, 센서네트워크, 무선 프로토콜, 소프트웨어 스트리밍, 미들웨어

박 홍 재

e-mail : hjpark0@kookmin.ac.kr

2006년 국민대학교 컴퓨터학부 컴퓨터과학/응용 전공(학사)

2006년~현재 국민대학교 일반대학원 전산과학전공(석사)

관심분야 : USN, 네트워크, 임베디드

한 왕 원

e-mail : wwhan@kookmin.ac.kr

2006년 국민대학교 컴퓨터학부 컴퓨터과학/응용 전공(학사)

2006년~현재 국민대학교 일반대학원 전산과학전공(석사과정)

관심분야 : 임베디드, 무선 프로토콜, 미들웨어

최 완

e-mail : wchoi@etri.re.kr

1981년 경북대 전자공학과 전산학(학사)

1983년 KAIST 전산학과 전산학(석사)

1995년 KAIST 전산학과 전산학(박사)

1985년 ETRI TDX10/ATM 전전자 교환기 개발 책임연구원(과책)

1997년 ETRI ATM 교환기 시스템 개발 책임연구원(실장)

1998년 ETRI 실시간 DBMS, OS, 컴파일러개발 사업 책임자

2001년 ETRI 실시간 DBMS 개발 팀장

2004년~현재 ETRI 온디맨드 서비스 연구팀 팀장

관심분야 : DBMS, ASP, 네트워크

허 성 진

e-mail : sjheo@etri.re.kr

1990년 경북대학교 컴퓨터공학과 졸업(학사)

1992년 경북대학교 대학원 컴퓨터공학과(석사)

1999년 경북대학교 대학원 컴퓨터공학과(박사)

1999년~2001년 창신대학 인터넷공학과 전임교수

2001년~현재 한국전자통신연구원 온디맨드서비스연구팀 선임연구원

관심분야 : ASP, 서버, 통신망, VoD 등