애플리케이션 공유 및 데이터 접근 최적화를 위한 씬-클라이언트 프레임워크 설계

(Design of Thin-Client Framework for Application Sharing & Optimization of Data Access)

송 민 규*
(Min-Gyu Song)

요 약 본 논문에서는 인터넷 상에서 애플리케이션 공유과 데이터 접근을 수행할 수 있는 씬-클라이언트 프레임워크를 설계할 것이며, 관련 기술로 X 윈도우 시스템, 가상 서버, CODA 파일 시스템, MPI(Message Passing Interface)를 활용하고자 한다. 우리는 네트워크 연결이 중단되더라도 서버 상에서 실행되던 애플리케이션을 로컬 상에서 실행할 수 있음은 물론 서버 상의 작업 수행으로 생성된 데이터에 클라이언트가 최적으로 접근할 수 있는 씬-클라이언트 프레임워크를 제안하고자 한다. 또한 네트워크가 복원되었을 때 로컬 상의 작업 내역이 서버에 효과적으로 반영될 수 있어야 할 것이다. 이러한 씬-클라이언트 프레임워크를 설계하기 위하여 본 논문에서는 기존의 시스템에 분산 Pseudo 서버, CODA 파일 시스템 기술을 접목시킬 것이며, 보다 효율적인 작업 수행, 관리를 위해 MPI를 활용할 것이다. 이를 통하여 네트워크 독립적인 씬-클라이언트 작업 환경을 구축할 수 있고 서버의 병목현상을 지양함으로써 다수의 사용자에게 확장성 있는 애플리케이션 서비스를 제공할 수 있다. 본 논문에서는 이를 구현함에 있어 기반이 되는 씬-클라이언트 프레임워크의 설계 방안에 대해 논의하고자 한다.

핵심주제어 : 씬-클라이언트, 서버 기반 컴퓨팅, X11, Pseudo Server, CODA File System, MPI(Message Passing Interface)

Abstract In this paper, we design thin-client framework capable of application sharing & data access on the Internet, and apply related skills, such as X windows system, pseudo server, CODA file system, MPI(Message Passing Interface). We suggest a framework for the thin client to access data produced by working on a server optimally as well as to run server side application, even in the case of network down. Additionally, it needed to reflect all local computing changes to remote server when network is restored. To design thin client framework with these characteristics, in this paper, we apply distributed pseudo server and CODA file system to our framework, also utilize MPI for the purpose of more efficient computing & management. It allows for implementation of network independent computing environment of thin client, also provide scalable application service to numerous user through the elimination of bottleneck on caused by server overload. In this paper, we discuss the implementing method of thin client framework in detail.

Key Words : Thin-Client, SBC(Server Based Computing), X11, Pseudo Server, CODA File System, MPI(Messaage Passing Interface)

* 한국천문연구원 기술개발연구본부

1. 서 론

네트워크 및 애플리케이션 기술의 발전에 따라 컴퓨팅 패러다임도 급속도로 변화하고 있다. 물리적으로 독립된 형태의 개별적으로 실행되던 컴퓨터 시스템은 네트워크 자체를 하나의 플랫폼으로 하여 거대한 가상 컴퓨터로 진화하고 있다. 뿐만 아니라 서버, 스토리지, 네트워크, 애플리케이션 등의 네트워크 상의 IT 자원을 각 사용자에게 서비스로 제공하는 클라우드 컴퓨팅이 새로운 컴퓨팅 방법으로 주목을 받고 있다[1][2]. 이러한 기술의 진보는 원격 시스템과 같은 네트워크 상의 리소스를 로컬에서 활용하는 것을 가능하게 하였다. 네트워크를 경유하여 타 시스템에 저장된 애플리케이션, 데이터의 호출이 가능하기에 사용자 시스템은 고사양일 필요가 없으며 경량화가 가능하다. 이러한 시스템은 씬-클라이언트 또는 네트워크 컴퓨터로 불리고 있으며 관련 기술로 Microsoft사의 터미널 서버스, Citrix사의 ICA(Independent Computing Architecture)가 폭넓게 활용되고 있다[3][8]. 하지만 이들 시스템은 단순하게 원격 서버의 디스플레이 이미지를 클라이언트로 전송한다는 한계를 가지고 있으며, 무엇보다 네트워크 장애 시 작업 수행이 불가능하다는 것이 가장 큰 단점으로 지적되었다.

본 논문에서는 이를 극복 및 보완할 수 있는 컴퓨팅 기법에 대해 알아보고자 한다. 우리가 지향하는 컴퓨팅 기술은 단순히 서버 상의 디스플레이 이미지를 수신하는 것을 넘어 실제 원격의 애플리케이션을 클라이언트에서 실행하는 것을 기본으로 한다. 또한 네트워크가 단절되더라도 그를 감지하여 클라이언트 상에서 작업을 지속할 수 있고 네트워크 복원 시, 이전의 세션을 유지할 수 있어야 한다. 이를 설계 및 구현하기 위한 기술로서 본 논문에서는 X 프로토콜, CODA파일 시스템, MPI(Message Passing Interface)를 활용할 것이다. X 프로토콜을 씬-클라이언트와 서버 시스템 간의 통신에 활용하여 로컬 상에서 서버 애플리케이션을 효과적으로 호출, 실행할 수 있는 시스템을 설계할 것이다[4][6][7]. 확장성 있는 클라이언트 시스템 지원 및 네트워크 비연결 모드에서의 작업 수행을 지원할 수 있도록 Pseudo 서버-클라이언트, CODA 파일 시스템을 시스템에 접목하였으며 이를 근간으로 애플리케이션 공유 및 데이터 접근에 있어서의 최적화 방안을 제안하고자 한다.

본 논문은 다음의 순서에 따라 구성된다. 먼저 씬-클라이언트의 기본적 원리, 동작 메커니즘과 기존 시스템의 문제점에 대해 2장에서 살펴볼 것이고 3장에서 에서는 기존 시스템의 한계를 극복할 수 있는 방안에 대해 서버 부하 분산, 네트워크 비연결 지원, 확장성 제고 측면을 중심으로 논하고자 한다. 4장에서 X 프로토콜, 분산 Pseudo 서버, CODA 파일 시스템, MPI 기술을 유기적으로 적용하여 다수의 씬-클라이언트에 애플리케이션을 서비스 할 수 있는 프레임워크를 실제적으로 설계하고자 하며 5장에서 설계된 시스템에 대한 장단점 및 향후 씬-클라이언트의 가능성을 진단하는 것으로 본 논문에 대한 결론을 맺고자 한다.

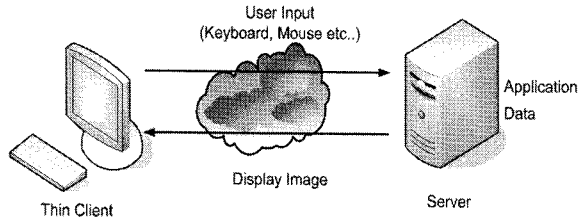
2. 씬-클라이언트 개요 및 개선 사항

씬-클라이언트는 기존 클라이언트/서버 모델에 서버 기반의 컴퓨팅(Server Based Computing)을 적용시킨 컴퓨팅 방식이다[9]. 작업 수행에 필요한 상당수의 애플리케이션, 데이터 등이 서버에 위치하기 때문에 사용자는 저사양의 시스템을 가지고도 로컬 상에서 작업을 효율적으로 수행하는 것이 가능하다. 씬-클라이언트를 활용함으로써 시스템의 구축, 유지 및 관리는 물론 활용에 있어 효율을 극대화시킬 수 있음은 물론 보안의 측면에 있어서도 보다 유연하게 대처할 수 있다[10]. 본 절에서는 먼저 씬-클라이언트의 근간이라 할 수 있는 서버 기반 컴퓨팅(Server Based Computing) 및 플러그-인 기반 컴퓨팅에 대해 살펴봄으로써 씬-클라이언트의 개념 및 원리 그리고 구성에 간략히 살펴보고자 한다. 나아가 씬-클라이언트 활용을 통해 얻을 수 있는 장점 및 효율성에 대해 알아보고자 한다.

2.1 서버 기반 컴퓨팅 개요

현재 컴퓨터 시스템은 CPU, 메모리, 하드디스크 등의 하드웨어는 물론 OS, 애플리케이션과 같은 소프트웨어 발전에 힘입어 갈수록 고사양-고급화되어가고 있다. 이에 따라 일반 가정을 비롯한 사무실, 연구실에서 활용되는 PC의 경우 서버로 활용해도 부족함이 없을 정도다. 더욱이 문서, 인터넷 검색, 메일 작성이 상당 비중을 차지하는 일반 업무용의 경우 현재의 PC(Personal Computer) 시스템은 오버스펙이라 하지 않을 수 없다. 로컬 시스템 상에서 상당한 작업이 이루어지는 성능에 비례하여 PC는 초기 도입은 물론 시스템 및 프로그램 업데이트, 유지 관리에 들어가는 비용이 급증할 수 밖에 없는 한계를 갖고있다. 또한 여러 장소에 산재해 있는 특성으로 일원화된 관리가 사실상 불가능하다. 이러한 저효율, 고비용을 극복하기 위하여 나타난 기술이 SBC(Server Based Computing)로서 데이터, 애플리케이션 등은 서버에 위치하게 된다[2][9].

SBC에서 작업 수행에 필요한 컴퓨팅 자원의 상당수는 서버에 위치하기 때문에 클라이언트를 저비용으로 구축하는 것이 가능하다. 뿐만 아니라 시스템을 구성하는 각종 하드웨어, 소프트웨어의 유지, 관리에 소비되는 비용을 최소화할 수 있다. 사용자는 자신의 시스템이 네트워크에만 연결되어 있으면 타 시스템에 접속하여 애플리케이션을 실행할 수 있고 원하는 결과를 얻을 수도 있다. 컴퓨팅 연산에 관련된 하드웨어 및 소프트웨어의 업그레이드는 대부분 서버에서 이루어지며 다수의 씬-클라이언트는 항상 최상의 컴퓨팅 서비스를 제공받을 수 있다[2]. 이러한 서버 기반 컴퓨팅의 개념을 그림으로 간략히 나타내면 아래 그림 1과 같다.

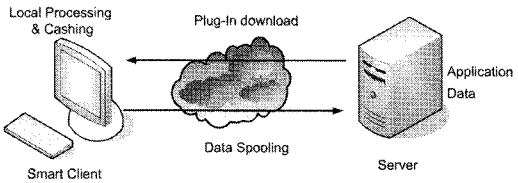


<그림 1> 서버 기반 컴퓨팅의 메커니즘

클라이언트는 마우스, 키보드 등의 사용자 입력을 서버로 전송하며 서버로부터 업데이트 된 디스플레이 정보를 수신한다. 작업 수행이 애플리케이션, 데이터가 저장된 서버에서 이루어지기에 사용자 시스템은 날씬하고 슬림한 형태로 구성되며 이는 씬-클라이언트로 지칭된다.

2.2 플러그-인 기반 컴퓨팅

서버 기반의 컴퓨팅에서 클라이언트의 입력신호는 서버로 전달된다. 모든 작업 및 연산 처리는 애플리케이션, 데이터가 저장된 서버에서 이루어지며 서버의 디스플레이 이미지가 클라이언트로 전달되었다. 입력 신호와 디스플레이 이미지가 송·수신되는 서버 기반 컴퓨팅과는 달리 플러그-인 기반의 컴퓨팅에서는 실행 가능한 애플리케이션이 클라이언트로 전달된다. 일반적으로 이는 웹 애플리케이션의 형태로서 실행되며 서버와의 통신에 기반하여 원하는 작업이 이루어진다. 모든 연산 처리가 전적으로 서버에 의존하는 서버 기반 컴퓨팅과 달리 플러그-인 기반의 컴퓨팅에서 클라이언트는 연산의 일부분을 담당하며 그를 위한 임의의 애플리케이션이 웹 브라우저와 상호작용하며 설치 운용된다. 이에 관련된 네트워크 기술로 ActiveX, 자바 애플릿, 플래시 등이 있으며 이를 통하여 클라이언트는 연산을 처리할 수 있는 애플리케이션을 웹 브라우저 상에서 실행하는 것이 가능하다[11]. 이러한 플러그-인 기술 기반의 컴퓨팅 개념을 그림으로 간략히 요약해보면 아래와 같다.



<그림 2> 플러그-인 기술을 활용한 네트워크 컴퓨팅

2.3 씬 클라이언트의 특징 및 개선 사항

씬-클라이언트는 네트워크 기술의 발전에 힘

입어 등장한 컴퓨팅 방식으로서 애플리케이션 실행 및 데이터 저장 등의 작업은 클라이언트가 서버에 접속한 후에 이루어진다. 따라서 씬-클라이언트는 서버와의 네트워크 연결을 필요로 하며 작업 수행에 있어 안정적인 네트워크 확보는 선택이 아닌 필수 조건에 해당한다[12]. 씬-클라이언트 활용에 있어 이는 장점이자 가장 큰 단점으로 작용하며 본 논문에서는 이러한 그를 극복할 수 있는 세부 방안에 대해 기술할 것이다.

네트워크 중심의 씬-클라이언트를 활용함으로써 얻을 수 있는 장점에는 여러 가지가 있지만 가장 대표적인 두 가지로 비용절감, 보안 강화를 들 수 있다[10]. 씬-클라이언트의 경우 서버 상에서의 연산 수행으로 컴퓨터가 고사양일 필요가 없다. 뿐만 아니라 HDD, CD-ROM 대용량 저장장치 또한 필요치 않다. 따라서 보다 저렴한 비용으로 시스템을 구축하는 것이 가능하며 초기 구매 뿐만 아니라 유지, 관리에 있어서 상당한 이점을 갖고 있다. 클라이언트/서버 컴퓨팅과 같은 기존 방식의 경우 각 PC에 대하여 소프트웨어 설치, 업그레이드, 프로그램 설정 등이 이루어져야 하였다. 하지만 씬-클라이언트는 기본적으로 애플리케이션과 데이터가 서버에 저장, 실행되며 클라이언트가 서버에 접속하여 작업을 수행하는 서버 기반 컴퓨팅이기 때문에 상기 작업은 상당부분 서버에서 수행하면 되며 보다 적은 비용, 노력으로 시스템을 운용하는 것이 가능하다[2].

이러한 효과는 보안의 경우도 마찬가지라 할 수 있다. 씬-클라이언트는 별도의 대용량 데이터 저장장치가 없기에 사용자 시스템 상에서 해킹을 근본적으로 차단할 수 있다. 상당량의 데이터는 애플리케이션과 함께 서버에 위치하며 오직 인가된 사용자만이 접근 가능하다. 따라서 관리자는 서버의 보안을 강화함으로써 네트워크 상에서 해킹, 데이터 유출, 바이러스 감염 등을 효율적으로 차단할 수 있으며 단일 시스템 상에서 각 사용자의 데이터에 대한 일원화된 관리를 구현할 수 있다[9]. 이러한 특성들로 많은 회사와 기업에서 씬-클라이언트 기반으로 네트워크를 구축하여 상당한 효과를 얻고 있지만 그 활용에 있어 씬-클라이언트의 단점 또한 크게 부각되는 상황이다. 서버 상에 데이터와 애플리케이션이 저장되어 실행되는 특성으로 네트워크 단절 시 작업이 사실상 불가능하다는 것은 씬-클라이언트의 가장 큰 단점으로 항상 언급되고 있다[12]. 이는 씬-클라이언트의 확산 및 성능에 가장 큰 영향을 주는 특성으로서 사용자 QoS의 측면에서 반드시 개선되어야 할 것이다. 또한 씬-클라이언트와 서버 간의 입력 신호, 디스플레이 정보 등의 빈번한 데이터 전송은 네트워크 병목 현상은 물론, 실제 애플리케이션의 활용에 있어 크나큰 오버헤드를 유발하는 가장 큰 원인으로 작용한다. 이와 같은 씬-클라이언트의 한계 및 활용에 있어서의 문제점을 간략히 분류해 보면 아래와 같다.

- 네트워크 종속적인 작업 수행

작업 수행에 필요한 모든 애플리케이션과 데이터가 서버에 저장되기에, 씬-클라이언트는 네트워크를 통해 해당 서버에 접속하여 작업을 수행한다. 이는 네트워크 장애 시, 씬-클라이언트 상에서 작업 수행이 크나큰 악영향을 받는다는 것을 의미한다. 더욱이 네트워크가 단절되는 상황이라도 발생하였을 때, 사용자의 작업 수행은 원천적으로 불가능함에 따라 이를 극복할 수 있는 방안이 마련되어져야 할 것이다.

- 씬-클라이언트 자체에서의 작업 불능

기존 씬-클라이언트의 경우에는 자체 시스템 상에서 데이터 접근, 애플리케이션 실행을 지원하지 않았다. 이러한 특성으로 사용자는 네트워크 장애 시 작업을 진행하는 것이 사실상 불가능하였고 안정적인 네트워크가 확보될 때까지 업무는 중단될 수 밖에 없었다. 사용자가 위치하는 모든 지역에 네트워크 연결을 보장할 수 없기에 네트워크 연결 독립적인 작업 수행을 위한 방안 마련이 이루어져야 할 것이다.

- 네트워크 트래픽 유발 및 빈번한 라운드 트립

씬-클라이언트는 작업이 실제로 실행되는 서버 시스템의 디스플레이를 전송받는다. 디스플레이 이미지가 서버로부터 씬-클라이언트로 전송되는 일련의 과정에서 상당한 네트워크 트래

픽이 유발된다. 무엇보다 클라이언트와 서버 간의 라운드 트립으로 인한 병목현상은 지연 발생의 가장 큰 요인으로 작용한다[13]. 보다 적은 대역폭으로 신속히 데이터를 전달할 수 있는 메커니즘이 구현되어야 하며, 이를 통하여 높은 QoS의 씬-클라이언트 시스템을 구축하는 것이 가능하다.

2.4 개선 사항

기존 씬-클라이언트는 오직 키보드, 마우스 등의 입력과 서버 디스플레이 출력만을 처리할 뿐 시스템 자체에 디스크가 없었고 그를 활용하려는 방안도 전무하였다. 모든 데이터와 애플리케이션은 서버 상에 저장되었고 작업 또한 전적으로 서버 상에서 이뤄지는 형태였다. 따라서 네트워크 단절 시, 서버 상의 애플리케이션, 데이터 접근은 원천적으로 차단되었고 작업 수행은 중단될 수 밖에 없었다. 네트워크 연결 상태에 따라 사용자 시스템이 무용지물이 되기도 하는 이러한 특성은 그동안 씬-클라이언트 활용에 있어 가장 큰 단점이자 한계로 지적되었으며, 네트워크 종속적인 이러한 특성으로 작업 능률이 현격히 저하되는 결과를 초래하였다.

본 논문에서는 이를 개선하기 위한 방안으로 임의 파일 시스템 기반의 스토리지 활용을 제안하고자 한다. 씬-클라이언트가 위치한 로컬 네트워크 상의 스토리지에 기본적인 애플리케이션이 설치되어 있다면 네트워크 단절 시, 중단된 작업을 이어서 진행하는 것이 가능하게 된다. 서버 애플리케이션 객체의 복사본이 클라이언트에 저장될 수 있다면 네트워크가 단절된 곳에서도 작업을 수행할 수 있음은 물론 이후 네트워크가 다시 복원되었을 때, 씬-클라이언트에서 수행하던 데이터 파일이 서버로 전달되어 서버 상에서 기존의 작업을 연속적으로 진행하는 것이 가능하다. 더불어 클라이언트의 작업 내역이 서버로 전달되어 재통합 과정을 거쳐 클라이언트 객체와 서버 객체가 동기화될 수 있다면 사용자는 씬-클라이언트와 서버가 일체화된 네트워크 작업 환경에서 보다 효율적인 작업 수행이 가능할 것이다. 네트워크 상태에 따라 유연한 이러한 메커니즘을 통해 사용자는 네트워크 장애 또는 이동 환경에서도 일관된 컴퓨팅 구현 및 세션 유지가 가능하다.

기존 씬-클라이언트에서 네트워크 단절 시, 애플리케이션 실행 및 데이터 접근이 원천적으로 불가능하였던 것은 모든 애플리케이션, 데이터가 서버에 위치하는 특성과 연관된다. 씬-클라이언트 상에서 그에 대한 복사본을 가질 수 있다면 네트워크 연결에 큰 영향을 받지 않고 작업을 수행하는 것이 가능할 것이다. 서버 상에서 실행되는 애플리케이션과 그로 인한 데이터를 클라이언트에서 임의의 형태로 보관해야 할 필요가 있으며 본 논문에서를 그러한 메커니즘을 기반으로 한 프레임워크를 설계할 것이다.

더불어 씬-클라이언트와 서버 간 다량의 데이터 전달 및 빈번한 라운드 트립 억제를 통하여 보다 안정적인 애플리케이션 실행을 구현할 수 있어야 하며 이를 구현하기 위한 분산 컴퓨팅 기법으로 가상 서버-가상 클라이언트(Pseudo Server - Pseudo Client)와 MPI(Message Passing Interface)를 활용하고자 한다[5][7].

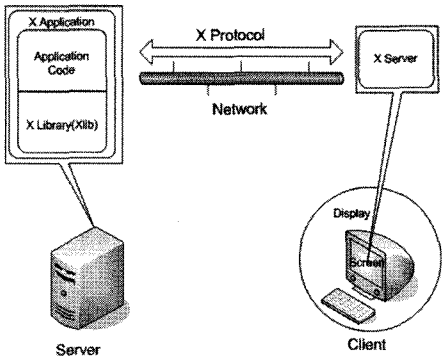
3. 시스템 설계 방안

본 장에서는 이전 장에서 논의되었던 씬-클라이언트 시스템의 문제점을 극복할 수 있는 개선 사항을 실제 씬-클라이언트 시스템에 어떻게 적용할 지에 대해 알아보기로 한다. 그를 구현하기 위한 기반 기술로서 본 논문에서는 X 프로토콜 데이터의 네트워크 분산, 네트워크 장애 시 캐시 데이터 활용, 애플리케이션 공유에 있어서의 확장성을 위한 미들웨어 활용을 제안하며 그 활용 방안에 대해 세부적으로 논의하기로 한다.

3.1 X 윈도우 시스템 적용

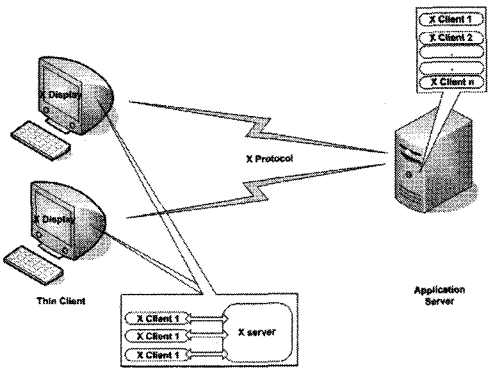
씬-클라이언트는 네트워크를 통하여 서버 상에 저장된 애플리케이션을 호출하는 방식으로 동작하며 이는 원격에 위치한 서버 애플리케이션을 호출하는 X 윈도우 시스템과 동일한 메커니즘

에 해당한다. 본 논문에서는 이러한 특성에 착안하여 씬-클라이언트 설계에 X 윈도우 시스템을 활용하고자 한다. X 윈도우 시스템 X11은 네트워크 지향의, 하드웨어 독립적인 그래픽 윈도우 시스템이며 그 기본 아키텍처를 그림으로 나타내면 그림 3과 같다[4][5].



<그림 3> X 윈도우즈 시스템 클라이언트/서버 아키텍처

X 윈도우 시스템 아키텍처에서 서버에서 실행되는 애플리케이션은 X 애플리케이션 또는 X 클라이언트로 불린다. 반면 클라이언트에서 그를 호출하며 활용하는 프로그램은 X 서버에 해당한다. X 서버는 애플리케이션 출력을 그래픽 형태로 사용자에게 도시할 뿐만 아니라 마우스, 키보드로부터 입력된 신호를 X 애플리케이션으로 전달하는 역할을 한다. 클라이언트와 서버 간의 통신은 X 프로토콜을 기반으로 수행됨에 따라 하나의 X 서버에서 여러 X 애플리케이션을 실행하는 것이 가능하다. 현재 씬-클라이언트 메커니즘 구현과 관련하여 널리 활용되는 프로토콜로 VNC(Virtual Network Computing), ICA(Independent Computing Architecture), RDP(Remote Display Protocol) 등을 들 수 있다[8]. 이들은 서버 상의 디스플레이 이미지를 원격의 씬-클라이언트에게 전달하는 방식을 사용하며 이로 인해 그래픽 이미지 전송에 상당한 대역폭이 소요된다. 또한 사용자가 서버 상의 애플리케이션을 직접 호출하는 것이 불가능하다는 점은 실제 애플리케이션을 호출함에 있어 크나큰 불편함으로 작용한다. 본 논문에서는 이러한 서버 디스플레이 전송을 통한 원격 애플리케이션 호출을 극복함에 있어 X 윈도우 시스템 적용을 제안하였으며 그를 통해 씬-클라이언트 사용자가 손쉽게 서버 상의 애플리케이션을 호출할 수 있도록 하였다. 이러한 X 프로토콜 기반의 씬-클라이언트를 설계하기 위한 기본 구성을 간략히 그림으로 나타내면 다음과 같다.



<그림 4> X 기반 씬-클라이언트 개념도

3.2 분산 Pseudo 서버 - 클라이언트 활용

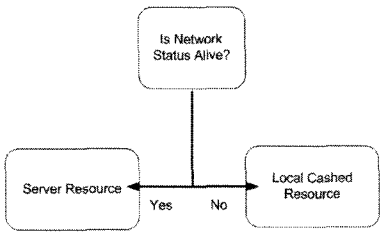
위 그림에서 알 수 있듯이 서버(X 클라이언트 또는 X 애플리케이션)로부터 전송된 디스플레이 요청은 결국 씬-클라이언트(X 디스플레이 또는 X 서버)에서 디스플레이 이미지로 구현되며, 이를 기반으로 씬-클라이언트 사용자는 원격의 애플리케이션을 실행하는 것이 가능하다. 로컬 사용자가 활용하기 원하는 서버 상의 X 애플리케이션은 X 프로토콜을 기반으로 씬-클라이언트의 X 서버와 통신하며 이를 근간으로 서버는 네트워크 상에서 애플리케이션을 공유하게 된다. 이 과정에서 서버가 처리해야 할 씬-클라이언트 수가 제한적이라면 X 애플리케이션은 성능 및 활용에 있어 아무런 문제가 없다. 하지만 서버(X 애플리케이션)에 연결된 씬-클라이언트의 개수가 증가하게 되면 시스템 병목 현상이 나타나며 그는 안정적인 세션을 유지하는 것을 매우 어렵게 한다. 무엇보다 X 프로토콜은 내부 네트워크 상에서 활용되기 위한 용도로 개발되어진 것이지 인터넷으로 대표되는 WAN에 최적화된 형태가 아니다[13].

이에서 알 수 있듯이 우리의 씬-클라이언트 프레임워크에서 기존의 X 프로토콜을 그대로 활용하는 것은 여러 문제를 발생시킬 가능성이 있다. 이러한 단점을 극복하기 위해서는 성능 및 안정성 측면에서 서버 상의 X 애플리케이션이 다수의 사용자 요청을 처리할 수 있어야 한다. 뿐만 아니라 서버가 X 애플리케이션의 요청을 처리하는 과정에서 병목현상이 발생하는 것을 억제할 수 있어야 한다. X 애플리케이션은 로컬 상에서 애플리케이션 실행 뿐만 아니라 원격에 위치한 다수 X 서버와 통신을 수행한다. 실행되는 애플리케이션 및 씬-클라이언트 개수가 늘어남에 따라 서버에 위치한 X 애플리케이션의 부담은 가중될 수 밖에 없으며 본 논문에서는 이러한 한계를 극복하기 위하여 분산 Pseudo 서버를 구성하고자 한다. 씬-클라이언트 아키텍처에 분산 Pseudo 아키텍처를 구성함으로써 애플리케이션 로드와 통신 로드를 분리할 수 있음은 물론, X 서버에 대한 효율적인 X 프로토콜 멀티플렉싱을 통해 안정적인 X 기반 씬-클라이언트 운용이 가능하다.

기존 시스템은 X 애플리케이션으로부터 네트워크 상의 다수 X 서버로 전송되는 X 요청에 대한 처리로 Pseudo 서버의 성능 저하는 물론, 병목 현상이 발생하기도 하였다. Pseudo 서버를 세분화하여 네트워크 상에 Pseudo 클라이언트를 분산배치 함으로 Pseudo 서버의 문제점을 극복할 수 있으며 보다 효율적이고 안정적인 X 애플리케이션 - X 서버 통신이 이루어질 수 있도록 하였다. 이를 위해 Pseudo 서버 - 클라이언트로 분산 Pseudo 서버를 세분화하였으며 효율적인 통신이 이루어질 수 있도록 Pseudo 서버, Pseudo 클라이언트는 각각 X 애플리케이션, X 서버와 동일한 네트워크 상에 위치하도록 설계하였다.

3.3 CODA 파일 시스템 기반의 네트워크 비연결 지원

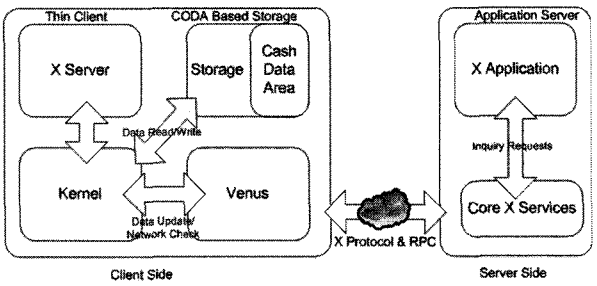
작업 수행에 필요한 모든 애플리케이션, 데이터가 서버에 위치하는 기존의 씬-클라이언트 시스템 구성은 실제 활용에 있어 많은 문제를 야기 시킨다. 가령 네트워크 대역폭 감소, 지연 등의 장애 발생 시 씬-클라이이언트에서 서버 애플리케이션과 데이터를 원활히 호출할 수 없으며, 네트워크 단절 시에는 서버 접근 차단으로 작업 수행이 원천적으로 불가능하였다. 이를 극복하기 위해서는 네트워크 연결 유무에 따라 동작 모드가 두 가지로 세분화될 필요가 있다. 씬-클라이언트가 네트워크에 연결되었을 때에는 서버 상의 애플리케이션, 데이터를 활용하다 네트워크가 단절되면 씬-클라이언트 자체에서 애플리케이션, 데이터를 처리할 수 있어야 하며 그에 대한 동작 모드를 간략히 그림으로 나타내면 아래와 같다.



<그림 5> 네트워크 상태에 따른 씬-클라이언트 동작 모드

네트워크 비연결의 상태에서도 애플리케이션 호출 및 데이터 접근이 가능하여야 한다. 본 논문에서는 이를 위하여 네트워크 단절 상황 발생 시, 로컬 상의 애플리케이션, 데이터에 접근할 수 있는 공유 데이터 저장장치 시스템을 로컬 네트워크 상에 구성할 것이며 그를 위한 방법으로 CODA 파일 시스템을 활용하고자 한다. CODA 파일 시스템은 현재 리눅스 커널과 함께 제공되는 오픈 소스 기반의 분산 파일 시스템으로 AFS(Andrew File System)와 비슷한 시스템을 유지하지만 차별화된 특성으로 서버 측 복제, 부분적인 네트워크 장애 시의 연속 연산, 확장성 및 대역폭 조정 기능을 제공한다[6]. CODA 파일 시스템 기반의 씬-클라이언트 시스템에서 사용자가 위치한 씬-클라이언트는 CODA 클라이언트에, 애플리케이션이 실행되는 서버는 CODA 서버에 해당한다. CODA 서버에 저장된 애플리케이션, 데이터는 캐시형태로 로컬 네트워크 상의 CODA 클라이언트에 저장된

다. 따라서 네트워크 연결이 단절되더라도 클라이언트는 로컬 네트워크 상에 캐시형태로 저장된 애플리케이션, 데이터를 호출함으로써 비연결 모드로 작업을 진행하는 것이 가능하다. CODA 파일 시스템 기반 네트워크 상에서 씬-클라이언트 시스템의 구성 및 동작 메커니즘을 도시해도면 그림 6과 같다.



<그림 6> CODA 기반의 네트워크 환경에서 씬-클라이언트 시스템의 구성

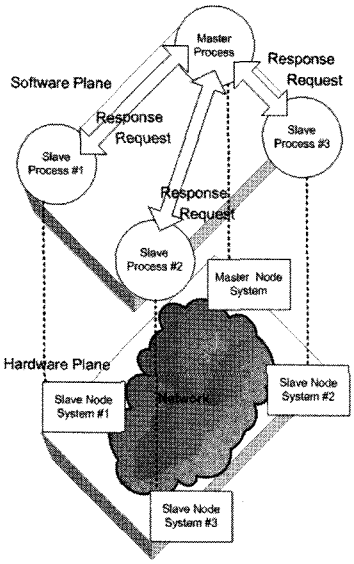
위 그림은 CODA 파일 시스템 기반의 네트워크 환경에서 씬-클라이언트가 서버 측 X 애플리케이션을 호출하는 메커니즘을 보여준다. X 서버의 마우스, 키보드 등의 장치로부터 입력신호가 들어오면 이는 CODA 캐시 관리자라 불리는 Venus에게 전달된다. 씬-클라이언트는 네트워크 연결 유무에 따라 로컬 네트워크 상의 캐시 활용 여부를 결정하며 전형적으로 이러한 상황은 호스트의 핸드오버 또는 네트워크 단절 시 발생한다. CODA를 활용한 연산은 이처럼 서버와의 통신이 단절되는 상황 발생 시 로컬 상에서 복원 가능한 파일 시스템을 제공하기 위함이다. CODA 클라이언트는 데이터 접근에 필요한 모든 정보를 CODA 서버로부터 캐싱하며 네트워크 비연결 상태에서 업데이트 된 로컬 네트워크 상의 파일은 네트워크 복원 시 서버에 반영된다. 씬-클라이언트 상의 사용자 입력, 이벤트, 요청에 대한 반환은 로컬 네트워크에서 CODA 파일 시스템 기반의 스토리지에 저장되며 네트워크 비연결 상태에서 이러한 모든 업데이트 정보는 클라이언트 변경 로그(Client Modification Log)라 불리는 CML의 형태로 스토리지에 저장된다. 따라서 CODA파일 시스템이 네트워크 비연결 모드로 전환되더라도 이후 서버에 재연결되었을 때 서버에 해당 요청이 전달되며 이러한 과정을 통해 클라이언트와 서버 간 최신의 상태로 세션을 유지하는 것이 가능하다.

3.4 확장성을 고려한 분산 컴퓨팅 인터페이스 적용

X 프로토콜 기반 씬-클라이언트 시스템에서 X 서버와 X 애플리케이션은 각각 씬-클라이언트와 서버 시스템 상에 위치한다. 씬-클라이언트에서 원활한 애플리케이션 활용이 이루어지기 위해서는 서버와 씬-클라이언트 간의 효율적인 통신 메커니즘이 구현되어야 한다. 기존의 X 윈도우 시스템은 X 애플리케이션에서 X 서버로 데이터 스트림을 전달함에 있어 유니캐스트 채널을 사용하는데 이는 확장성 면에서 두 가지 문제점을 갖는다. 그 중 첫 번째로 서버 시스템의 병목현상을 들 수 있는데 X 애플리케이션을 요청하는 씬-클라이언트가 일대일 형태이면 문제없겠으나 그것이 다수로 확장되면 X 프로토콜 데이터를 멀티플렉싱 하는 시스템의 부담은 그만큼 증폭될 수 밖에 없다. 실질적으로 서버 또는 동일 네트워크 상의 Pseudo 서버의 경우 다수 씬-클라이언트와의 통신 수행과 세션 관리로 인해 상당한 시스템 과부하가 유발되고 있다. 서버 애플리케이션을 활용함으로써 컴퓨팅 작업을 수행하는 대다수의 씬-클라이언트에게 이는 반드시 개선되어야 할 사항으로서 부하를 효율적으로 분산시킴으로서 이 문제점을 극복할 수 있다.

두 번째 문제는 X 애플리케이션을 활용하는 씬-클라이언트 사용자의 동적 가입-해제에 있어서의 난해함을 들 수 있다. 이는 X 애플리케이션이 실행되는 서버 상에서 X 프로토콜 데이터에 대한 수신-처리 메커니즘이 씬-클라이언트와의 세션 구현과 분리되지 않아 나타나는 현상이다. 이를 극복하기 위하여 본 논문에서는 X 윈도우 시스템 환경에서 애플리케이션 공유를 위한 분산 시스템 아키텍처를 제시하고자 한다. X 기반 애플리케이션을 네트워크 상에 공유시키는 프레임워크를 설계함에 있어 그를 다수의 씬-클라이언트로 확장시키기 위해 본 논문에서는

분산 컴퓨팅 인터페이스 MPI(Message Passing Interface)를 도입하였다. MPI는 우리의 씬-클라이언트 프레임워크에서 X 애플리케이션의 요청을 처리하여 로컬 X 서버에 반영시킴은 물론 씬-클라이언트-서버 간의 데이터 처리를 수행한다[7]. 본 논문에서 우리는 서버 상에서 X 애플리케이션 요청 처리 메커니즘과 씬-클라이언트와의 통신 메커니즘을 효과적으로 분산시킴에 있어 분산 컴퓨팅의 한 형태인 LAM/MPI를 활용하고자 하며 그에 대한 대략적인 개요를 그림으로 나타내면 아래와 같다.



<그림 7> MPI 구성 및 동작 메커니즘

MPI(Message Passing Interface)는 컴퓨터 클러스터(computer clusters)나 병렬 컴퓨터(parallel computers) 상에서 실행 가능한 프로그램 개발 기능을 제공하는 표준화된 API로서 본 논문에서는 각 네트워크에 공유 애플리케이션 활용에 필요한 각 기능을 수행할 수 있는 마스터 노드와 슬레이브 노드 지정하였다. 서버 측 네트워크 상에서 마스터 노드는 X 프로토콜 데이터 스트림에 대한 수신 및 처리를 수행하며 슬레이브 노드는 네트워크 상에서의 씬-클라이언트 세션 유지 및 안정적인 X 프로토콜 멀티플렉싱의 기능을 수행한다. 이를 통해 공유 애플리케이션 운용과 씬-클라이언트와의 저수준 통신 메커니즘을 분리할 수 있고, 네트워크 상에서 안정적인 씬-클라이언트 시스템 구축 및 운용이 가능하다.

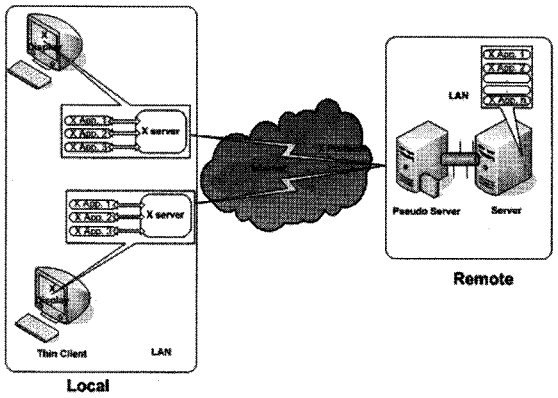
4. 확장성 있는 X 기반의 씬-클라이언트 시스템 설계

본 장에서는 지금까지 살펴본 씬-클라이언트 시스템의 문제점 및 개선방안을 기반으로 씬-클라이언트를 실제 활용함에 있어 확장성, 안정성을 배가시킬 수 있는 시스템을 설계하고자 한다. 그를 위해 이전 장에서 살펴본 X 프로토콜, Pseudo 서버-클라이언트, CODA 파일 시스템, MPI 기술을 적용할 것이며, 씬-클라이언트 시스템을 설계함에 있어 그를 유기적으로 결합하고자 한다.

씬-클라이언트는 네트워크를 통해 서버 상에 위치해 있는 애플리케이션을 실행하고, 그에 대한 결과를 얻을 수 있어야 한다. 그러한 시스템을 구현하기 위한 기본 형태로서 본 논문에서는 X 윈도우 시스템을, 프로토콜로는 X 프로토콜을 활용하고자 한다. 씬-클라이언트 시스템을 X 윈도우 시스템으로 구현함에 있어 X 애플리케이션, X 서버 두 가지 요소에 주목할 필요가 있다. X 애플리케이션은 서버 상에서 실행되는 애플리케이션에 해당하며 X 서버는 그를 호출하는 씬-클라이언트 콘솔에 해당한다.

하나의 서버는 네트워크 상에서 다수의 씬-클라이언트를 지원할 수 있어야 하며 그를 위한 서버 시스템의 확장성과 더불어 각 씬-클라이언트와의 안정적인 통신이 수반되어야 한다. 단일 씬-클라이언트와 서버로 구성되는 단순 네트워크 구조에서는 유니캐스트 통신 메커니즘을 X 서버, X 클라이언트에 그대로 적용할 수 있다. 하지만 절대다수의 시스템이 네트워크를 구성하며 상당량의 트래픽이 유발되는 현실에서 서버 애플리케이션을 공유하는 클라이언트 수는 급증할 수 밖에 없다. X 프로토콜에 기반하는 씬-클라이언트 - 서버 간 통신에 있어 확장성, 성능의 측면에서 가장 큰 걸림돌은 X 서버에 대한 X 애플리케이션의 요청 처리 및 각 씬-클라이언트에 대한 멀티플렉싱, 세션 유지라 할 수 있

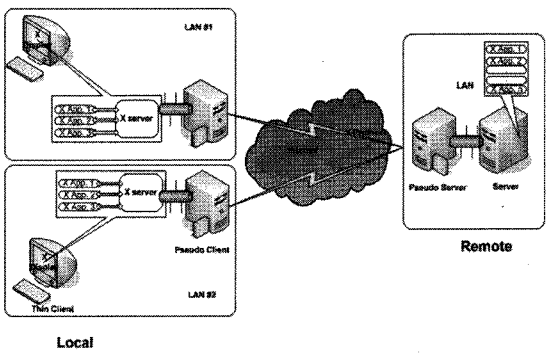
다. 하나의 서버에서 이를 처리하는 아키텍처는 시스템의 병목현상을 유발하며 결과적으로 서버 애플리케이션을 이용하는 타 씬-클라이언트에도 악영향을 초래한다. 이를 극복하기 위한 첫 시작으로 본 논문에서 활용한 방식은 Pseudo 서버이다. Pseudo 서버는 서버와 동일한 네트워크 또는 서버 시스템 자체에서 실행되는 가상의 시스템으로 X 애플리케이션의 요청을 수신하여 그를 씬-클라이언트(정확히 말하면 X 서버)로 전달하는 역할을 한다. 단순히 X 애플리케이션의 요청을 전달만 하는 것이 아니라 씬-클라이언트 수가 늘어났을 때 그를 멀티플렉싱하며 세션을 유지하는 기능도 수행하며 이에 대한 시스템 설계를 그림으로 도시해보면 다음과 같다.



<그림 8> Pseudo 서버를 통한 서버 측 부하 로드 밸런싱

이를 통해 X 애플리케이션이 실행되는 서버의 작업을 타 시스템으로 분배함은 물론, 성능 개선 효과 또한 얻을 수 있다. 하지만 이는 중앙 집중화된 Pseudo 서버 방식으로서 하나의 시스템에서 다수 씬-클라이언트와 멀티플렉싱을 통한 세션 유지를 지속해야 하는 것과 X 애플리케이션 요청을 처리해야 하는 구조는 크게 달라지지 않는다. 단일 서버 시스템과 마찬가지로 시스템의 성능 저하 및 불안정한 세션을 초래한다는 점에서 이러한 중앙 집중화된 Pseudo 서버 방식은 바람직한 해결책이라 할 수 없다. 단일 Pseudo 서버로 집중되는 X 애플리케이션 요청, X 서버의 응답에 대한 처리 및 씬-클라이언트와의 세션 유지 기능을 네트워크 상에 좀더 분산시킬 필요가 있으며 그를 위한 방법으로서 분산 Pseudo 서버 아키텍처를 활용하고자 한다.

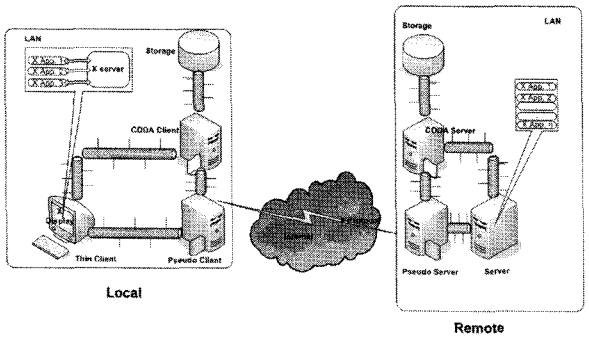
분산 Pseudo 서버 아키텍처는 서버 측의 Pseudo 서버, 클라이언트 측의 Pseudo 클라이언트 두 컴포넌트로 세분화되며 이전에는 단일 Pseudo 서버에서만 행해지던 X 애플리케이션의 요청 처리가 Pseudo 클라이언트로 분담된다. 이러한 분산 Pseudo 서버 아키텍처에서 X 애플리케이션으로부터 수신된 요청은 씬-클라이언트 상의 Pseudo 클라이언트에서 처리되며 X 서버의 디스플레이에 필요한 시퀀스 넘버, 아톰 넘버, 컬러 정보, 윈도우 좌표 그리고 다양한 식별자에 대한 매핑이 씬-클라이언트에서 구현된다. 하나의 Pseudo 서버가 아닌 X 애플리케이션을 필요로 하는 각 씬-클라이언트 상에서 개별적으로 해당 요청을 처리하는 이러한 방법을 통해 시스템 성능을 최적화시킬 수 있다. 뿐만 아니라 X 서버와 동일한 시스템 또는 네트워크 상에서 해당 기능이 수행되기에 보다 효율적으로 씬-클라이언트를 운용하는 것이 가능하다. X 애플리케이션 요청 처리 및 씬-클라이언트와의 데이터 송수신을 Pseudo 클라이언트에서 수행함으로써 보다 확장성 있고 안정적인 씬-클라이언트를 설계할 수 있으며, 그에 따라 본 논문에서는 씬-클라이언트, 서버가 위치하는 각 로컬 네트워크 상에 Pseudo 클라이언트- 서버를 함께 구축하고자 하며 그에 대한 시스템 구성을 그림으로 나타내면 그림 9와 같다.



<그림 9> 씬-클라이언트 네트워크 상의 Pseudo 클라이언트 구성

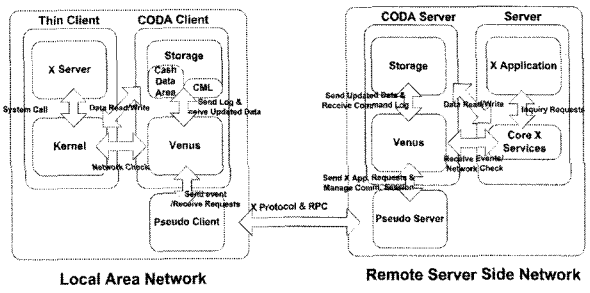
이와 더불어 네트워크 안정적이고 사용자에 최적화된 씬-클라이언트 구현을 위해서는 네트워크 비연결 상태에서도 작업을 지속할 수 있어야 한다. 씬-클라이언트에서 작업 수행에 필요한 애플리케이션, 데이터는 기본적으로 서버에 위치하기 때문에 네트워크 장애 또는 단절 시 작업 수행이 원천적으로 불가능한 경우가 발생하고 씬-클라이언트가 무력화되는 크나큰 문제점이 존재하였다. 이를 극복하기 위해서는 네트워크 불능의 상태에서도 씬-클라이언트에서 작업 수행이 가능해야 하며 애플리케이션, 데이터가 씬-클라이언트에 저장되어 있다면 그를 구현하는 것이 가능하다. 이는 전적으로 애플리케이션, 데이터를 저장하는 파일 시스템에 관련된 사안으로서, 본 논문에서는 네트워크 비연결 상태에서의 작업 수행을 가능하게 할 이러한 방법으로 씬-클라이언트 프레임워크에 CODA 파일 시스템을 적용하고자 한다.

CODA 파일 시스템은 Venus를 활용하여 서버의 파일을 캐시 형태로 로컬 상에 저장하며 궁극적으로는 네트워크 장애 시 복원 가능한 파일 시스템 제공을 그 목적으로 한다. 따라서 씬-클라이언트가 네트워크 단절되더라도 애플리케이션, 데이터가 로컬 상에 캐시 형태로 존재하기에 작업을 지속하는 것이 가능하다. 애플리케이션 및 데이터 처리에 있어 CODA 파일 시스템을 적용한 씬-클라이언트 시스템 구성을 나타내면 다음과 같다.



<그림 10> 네트워크 비연결 상태에서의 컴퓨팅 연산을 위한 CODA 시스템 설계

애플리케이션, 데이터 등의 정보가 씬-클라이언트 상에 저장되기에 HDD, USB 인터페이스에 기반한 임의의 스토리지가 장착되어 있어야 한다. 씬-클라이언트 상의 스토리지에 미리 설치되어져 있는 애플리케이션은 네트워크 단절 시 씬-클라이언트에 의해 호출되어진다. 서버 상의 애플리케이션 또는 로컬 스토리지 상의 애플리케이션 둘 중 어느 쪽을 호출할지는 네트워크 연결 유무에 달려 있다. 본 논문에서는 그를 판단하는 하나의 모듈로서 CODA 파일 시스템의 Venus를 활용할 것이며 스토리지, 시스템 커널, Venus로 파일 저장 메커니즘이 동작하는 씬-클라이언트 시스템을 구성하고자 한다. 씬-클라이언트 상에서 사용자가 X 서버를 통해 서버 상의 X 애플리케이션에 제어 신호를 전달 할 때 Venus를 통해 네트워크 연결 유무를 판단한 후, 애플리케이션 호출 또는 데이터 캐싱 절차가 이루어진다. 서버로부터 데이터 획득은 원격 절차 호출(Remote Procedure Call)을 통해 이루어지며 해당 파일은 스토리지의 캐시 영역(예: /usr/coda/venus/cache)에 컨테이너 파일로서 저장된다. 이러한 CODA 파일 시스템 기술을 적용함으로써 실행 가능한 씬-클라이언트 시스템의 메커니즘을 다이어그램으로 나타내면 아래 그림 11과 같다.



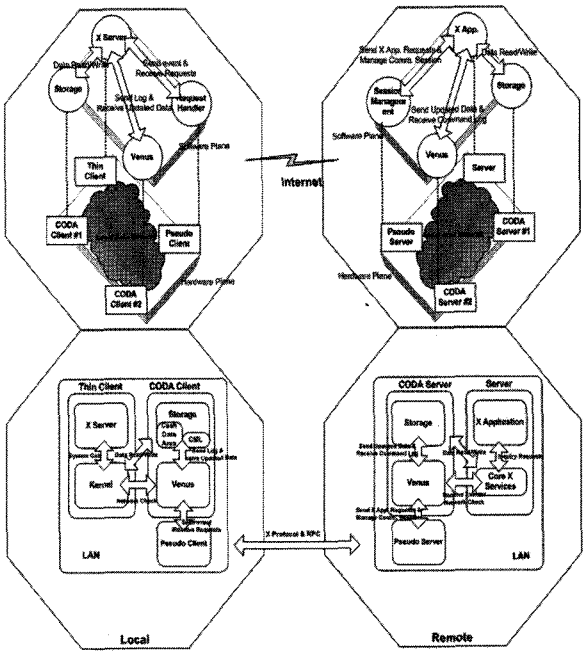
<그림 11> CODA 기술 기반 씬-클라이언트 시스템의 동작 메커니즘

씬-클라이언트 프레임워크를 최적화하기 위한 방안으로 프로토콜 측면에서는 X11과 RPC를, 하드웨어 측면으로는 스토리지, 분산 Pseudo 서버를 활용하였다. 하지만 실제 네트워크 상에서 X 프로토콜에 대한 요청 및 송수신 데이터 처리, 그리고 양 시스템 간의 세션 유지 등의 작업

은 Pseudo 시스템을 통해 구현되며 보다 안정적으로 시스템 운용 및 성능 향상의 측면에서 분산 시스템을 구축할 필요가 있다. 이와 관련 본 논문에서 제안한 컴퓨팅 방식은 LAM/MPI이다. 이미 이전 장에서 설명하였듯이 MPI(Message Passing Interface)는 컴퓨터 클러스터(computer clusters)나 병렬 컴퓨터(parallel computers) 상에서 실행 가능한 프로그램 개발 기능을 제공하는 표준화된 인터페이스로 동일 네트워크 상에서 다수의 컴퓨터를 지정하여 공유 애플리케이션 및 통신 세션을 분산 처리하려는 본 논문의 취지에 부합한다. 서버 상의 X 애플리케이션으로부터 씬-클라이언트 상의 X 서버로 향하는 요청과 두 시스템 간에 송수신되는 데이터를 처리함에 있어 본 논문에서는 MPI 호출을 활용하고자 한다.

LAM/MPI를 활용한 씬-클라이언트 프레임워크에서 씬-클라이언트, 서버가 위치한 네트워크에는 X 애플리케이션의 요청 처리 및 데이터 통신 수행을 위하여 각각 Pseudo 서버, Pseudo 클라이언트가 구동된다. LAM/MPI 시스템 라이에서 서버와 씬-클라이언트에서 실행되는 X 애플리케이션, X 서버는 마스터노드로서 동작한다. 반면 Pseudo 서버-클라이언트, CODA 서버-클라이언트 시스템은 슬레이브 노드로서 각 네트워크에서 애플리케이션 공유 및 데이터 접근을 실제적으로 수행하는 역할을 한다. 이의 활용을 통해 서버의 X 애플리케이션은 씬-클라이언트 X 서버와의 통신에 있어 보다 효과적으로 데이터와 애플리케이션을 처리할 수 있으며 LAM/MPI 구조를 통해 보다 넓은 네트워크 범위에서 확장성을 가지고 애플리케이션이 공유될 수 있도록 한다. 마찬가지로 씬-클라이언트 상의 X 서버와 동일한 내부 네트워크 상에 Pseudo 클라이언트, CODA 클라이언트를 슬레이브 노드로 지정하여 본 논문에서는 X 애플리케이션 메시지 처리-실행에 있어 보다 안정적이고 최적화된 씬-클라이언트 메커니즘을 구현할 수 있도록 하였다.

지금까지 네트워크 성능에 최적화되고 다수의 씬-클라이언트에게 애플리케이션을 공유할 수 있는 씬-클라이언트 프레임워크를 설계함에 있어 필요한 기술 및 적용 방안에 대해 논의하였다. X 프로토콜, Pseudo 서버-클라이언트, CODA 기반의 스토리지, MPI 인터페이스를 우리의 씬-클라이언트 아키텍처에 순차적으로 적용하는 과정에서 그 가능성 및 효과에 대해 논의할 수 있었으며 이를 통해 제안된 씬-클라이언트 프레임워크를 나타내면 그림 12와 같다.



<그림 12> 제안된 씬-클라이언트 시스템 프레임워크

5. 결 론

씬-클라이언트는 네트워크를 경유하여 원격 서버에 위치한 애플리케이션, 데이터를 호출하는 컴퓨팅 방식이다. 애플리케이션, 데이터가 로컬에 저장되는 기존 클라이언트-서버 모델은 네트워크 기술의 발전에 힘입어 씬-클라이언트로 진화하고 있으며, 이제 사용자는 네트워크에 연결되어 있다면 언제든지 네트워크 상의 리소스를 활용함으로써 원하는 업무를 수행하는 것이 가능하게 되었다.

본 논문에서 우리는 이러한 씬-클라이언트를 보다 안정적이고 효율적으로 구현 및 활용할 수

있는 시스템 프레임워크 설계에 대해 살펴보았다. 또한 단순히 하나의 씬-클라이언트와 서버로 구성되는 개념적 활용을 넘어 네트워크 상의 수많은 씬-클라이언트에 실제적으로 애플리케이션을 제공할 수 있는 프레임워크 설계 방안에 대해 알아보았다.

원격에 위치한 서버 상의 애플리케이션을 호출함에 있어 본 논문에서는 X 윈도우 시스템 활용을 기반으로 하였다. X 기반 씬-클라이언트 프레임워크의 두 핵심 요소에 해당하는 X 서버, X 애플리케이션은 각각 씬-클라이언트, 서버에서 실행되며 사용자는 X 서버를 콘솔로 하여 X 애플리케이션을 호출한다. 로컬 네트워크에 최적화된 X 프로토콜을 인터넷 상의 불특정 다수 씬-클라이언트와의 통신에 적용하기 위해 본 논문에서는 분산 Pseudo 서버, LAM/MPI 기법을 활용하였다. 또한 네트워크 비연결 상태에서도 일관된 작업 관리를 지원할 수 있도록 CODA 파일 시스템을 활용하였다. 씬-클라이언트 상의 X 서버에 대한 서버의 X 애플리케이션 요청과 함께 서버에 지속적으로 유입되는 데이터에 대한 처리는 서버의 병목 현상을 유발하는 가장 큰 원인이었다. 또한 다수의 씬-클라이언트에게 안정적으로 애플리케이션을 제공함에 있어 가장 큰 장애물이기도 하였다. 하지만 서버에 집중되는 이러한 작업을 각 시스템에 고루 분산시킴으로써 이러한 문제를 해결할 수 있으며 사용자 측면에서도 보다 안정적으로 서버 애플리케이션을 활용하는 것이 가능하다. 이를 위하여 본 논문에서는 씬-클라이언트와 서버 시스템이 위치한 네트워크 상에 각각 Pseudo 서버, Pseudo 클라이언트를 구축하는 방식으로 분산 Pseudo 서버 아키텍처를 설계하였고 그를 효율적으로 관리하기 위한 제어평면으로 LAM/MPI를 활용하였다. LAM/MPI는 한 네트워크 상에서 여러 운영체제로 작동되는 컴퓨터들을 위한 Local Area Multicomputer (LAM) MPI 프로그래밍 환경 및 개발 시스템으로서 본 논문에서 설계한 씬-클라이언트 아키텍처에서 씬-클라이언트와 서버는 마스터노드에 해당하며 Pseudo 서버-클라이언트는 슬레이브 노드로서 동작한다. LAM/MPI 구조 하에서 X 애플리케이션의 요청처리 및 X 서버에 대한 데이터 송수신은 슬레이브 노드를 통해 이루어지며 그에 대한 결과가 마스터 노드인 씬-클라이언트, 서버에 반영된다.

다수 씬-클라이언트에 대한 공유 애플리케이션 지원 및 보다 확장성 있는 서버 운용을 위한 상기 조치와 더불어 본 논문에서는 네트워크 상태 독립적인 씬-클라이언트 설계를 목표로 한다. 그를 위한 파일 시스템으로 본 논문에서는 CODA 파일 시스템을 채택하였으며 그를 LAM/MPI 클러스터에 적용하였다. CODA 파일 시스템의 Venus를 통해 서버 상의 데이터는 클러스터를 구성하는 다수 슬레이브 노드의 동일한 캐시영역에 컨테이너 파일로 저장된다. 뿐만 아니라 씬-클라이언트 상에서 행해진 파일에 대한 작업 내역은 네트워크 연결 유무에 관계없이 서버에 전송되어 서버의 파일 시스템을 업데이트 한다. 이러한 메커니즘을 기반으로 각 사용자는 씬-클라이언트에서 네트워크 상태에 독립적인 애플리케이션 실행, 파일 접근이 가능하게 된다. 또한 씬-클라이언트와 서버의 파일이 최신의 파일 상태를 유지함으로써 사용자는 시간, 거리에 관계없이 마치 서버 상에서 작업을 수행하는 것 같은 효과를 얻을 수 있다. 이는 본 논문에서 궁극적인 목표로 하는 애플리케이션 공유 및 데이터 접근 최적화를 위한 씬-클라이언트 프레임워크에 전적으로 부합하며 이를 통해 네트워크 상에서 다수의 사용자들이 보다 안정적으로 서버의 애플리케이션을 활용하는 것이 가능함을 확인하였다.

참 고 문 헌

[1] Aaron Weiss(2007). *Computing in the Clouds.* ACM. 2007. pp.16-25.

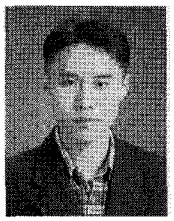
[2] Andrej Volchkov(2002). *Server-Based Computing Opportunities.* IEEE. pp.18-23.

[3] Niall Lynch(1999). *Supporting Disconnected Operation in Mobile CORBA.* pp.41-49.

[4] http://en.wikipedia.org/wiki/X_Window_System.

[5] Martin Mauve(1997). *Protocol Enhancement and Compression for X-Based Application*

Sharing. Univeresity of Mannheim. 1997. pp.23-28.
[6] http://www.coda.cs.cmu.edu/ljpaper/lj.html.
[7] *Message Passing Interface Forum. MPI: A Message-Passing Interface Standard.* 2003.
[8] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper (1998). *Virtual Network Computing.* Mobile Computing. pp.33-38.
[9] Joel Kanter(1999). *Understanding Thin-Client/Server Computing.* Microsoft.
[10] Jan Krikke(2004). *Thin Clients Get Second Chance in Emerging Markets.* IEEE CS pp.6-10.
[11] http://www.pulsewan.com/data101/thin_client_basics.htm.
[12] Sung-Seok Kang et al(2007). *Location Based Security Architecture Based on Smart Client Model For Mobile Environment.* ITNG'07.
[13] White Paper(2009). *The Next-Generation PC X Server. Attachmate.* pp.1-7.
[14] Edmund B. et al(2005). *Speculative Execution in a Distributed File System..* SOSP'05. pp.191-205.



송 민 규 (Min-Gyu Song)

- 정회원
- 2001년 2월 : 강원대학교 전기학과 (공학학사)
- 2003년 2월 : 강원대학교 전자공학과 (공학석사)
- 2002년 12월 ~ 현재 : 한국천문연구원 연구원
- 관심분야 : e-VLBI(Very Long Baseline Interferometry) , 클라우드 컴퓨팅, 웹 서비스, 초고속 네트워크, GRID