□특집□

클라이언트/서버 환경과 컴퓨팅 모델

문 현 주† 황 인 재†† 김 석 일†††

◈ 목 차 ◈

1. 서 론

클라이언트/서버 컴퓨팅은 분산처리의 한 형태로써 클라이언트가 필요한 요구사항을 서버에게 보내면 서버는 그 요구사항을 처리하여 결과를 클라이언트에게 되돌려주는 협동적 방식을 말한다. 클라이언트/서버 시스템에는 디스크의 파일이나 프린터와 같은 공유자원이 두 개 이상의 분리된 컴퓨터에 산재한다. 이러한 공유자원들을 제공하는 시스템을 서버라 부르며 이들은 클라이언트로부터의 서비스 요청을 받는데 사용된다. 분산처리 시스템이 시작되기 전의 가장 초보적 환경인 호스트 기반 처리에서는 더미(dumb) 터미널이 부착된 컴퓨터 시스템에서 모든 작업이 수행되었다. 다음 단계인 주종속(master-slave) 처리에서는 종속컴퓨터가 주컴퓨터에 연결되어 있고 종속컴퓨터는 주컴퓨터가 지시하는대로 응용 처리 관련 기능만을 수행하였다. 그러나 클라이언트/서버 시스템에서는 클라이언트가 데이터의 표현(presentation)과 응용 로직을 수행할 뿐만 아니라 서버의 데이터 관리기능의 일부까지 담당하므로써 클라이언트의 역할이 커지고 있는 추세이다.

클라이언트/서버 컴퓨팅은 근거리 통신망(LAN)에서 흔하게 볼 수 있는 공유자원처리가 자연스럽게 확장된 개념이라 할 수 있다. 근거리 통신망의 규모가 커짐에 따라 지원되는 워크스테이션의 수가 많아지고 이러한 워크스테이션들이 클라이언트가 되기도 한다. 이에 따라 파일 공유나 프린트 서비스와 같은 기본적인 기능 이외에 어떤 응용처리에서는 클라이언트 역할을 하는 워크스테이션에서 수행되는 응용 프로그램으로부터 발생하는 요구사항을 서버에 분산하게 되었다. 이 경우의 처리는 균등하지는 않지만 클라이언트와 서버에 분산되며 서비스 요청자인 클라이언트가 처리를 시작하고 부분적인 제어를 하게 된다. 따라서 주종속 처리와는 달리 클라이언트와 서버가 협동적으로 일을 수행하게 된다. Sybase나 microsoft

† 준회원 : 충북대학교 전자계산학과 박사과정
†† 정회원 · 충북대학교 컴퓨터교육과 조교수
††† 정회원 : 충북대학교 컴퓨터과학과 부교수

의 SQL 서버와 같은 데이터베이스 서버는 클라이언트/서버 컴퓨팅 환경의 좋은 예이다. 일반적으로 클라이언트/서버 컴퓨팅은 다음과 같은 장점을 갖는다.

- 클라이언트/서버 컴퓨팅은 데스크탑 컴퓨터의 계산 기술을 활용하여 저렴한 비용으로 중대형 컴퓨터에서나 가능하던 상당한 계산의 작업을 수행할 수 있다.
- 다양한 데이터 자원에 대한 동시 접근을 가능케 함으로써 의사결정 등 기업운영에 필요한 데이터 접근 시간을 줄일 수 있다.
- 다수의 사용자가 자원 및 컴퓨팅 도구를 공유하므로 가용한 연산 자원을 효율적으로 사용할 수 있다. 또한 새로운 소프트웨어의 추가 및 삭제에 관한 처리가 서버에서만 필요로 되므로 시스템의 유지 보수가 쉽고 호환성 문제를 줄일 수 있다.
- 클라이언트/서버 개발 환경에서는 표준화된 인터페이스를 통한 코딩을 모듈화 할 수 있으며 팀 단위의 프로그램 개발이 용이하다.

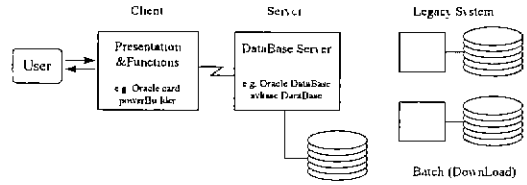
그러나 클라이언트/서버 컴퓨팅에는 다음과 같은 한계가 있다.

- 개발 환경 측면에서 보면 클라이언트/서버 컴퓨팅은 응용 프로그램 구현의 복잡성을 증대시키고 개발된 응용 프로그램의 검증이 어렵다. 더욱이 분산 클라이언트/서버 환경 구현을 위해서는 표준화된 middleware의 선정이 필수적이다.
- 클라이언트/서버 컴퓨팅은 클라이언트와 서버간에 많은 통신량을 수반하므로 네트워크 성능의 최적화를 위한 응용 프로그램이 설계되어야 하며 현재 LAN 환경을 구축하고 있는 ethernet의 한계를 고려해야 한다.
- 단일 머신과 달리 분산환경은 이를 관리할 수 있는 새로운 프로세스나 도구가 부족하다. 또한 이를 관리하기 위해서는 숙련된 인력이 요구되며 새로운 기술을 보급하기 위한 추가적인 비용이 요구된다.

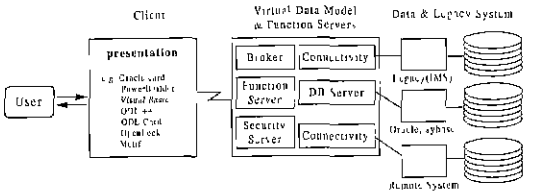
본 고에서는 클라이언트/서버 컴퓨팅 환경과 컴퓨팅 모델에 관하여 설명하고자 한다. 특히 클라이언트가 필요로하는 요구사항이 여러 개의 서버에 의하여 충족되어야 하는 경우, 클라이언트와 서버간의 역할 분담 및 상호동작에 관한 사항들을 자세히 설명하며 클라이언트/서버 환경과 관련된 소프트웨어의 발전 경향을 분석한다.

2. 클라이언트/서버 구조

최근에 설계된 클라이언트/서버 시스템들은 대부분 3층(three-tier)구조를 갖는다. 3층 구조에서는 (그림 1)과 같이 서버와 클라이언트 사이에 중간 단계의 에이전트를 둔다. 이러한 에이전트는 메인 프레임상의 응용 프로그램을 클라이언트/서버 환경에 적용시키기 위한 번역 서비스를 제공하거나 하나의 서버에 클라이언트로부터 오는 요청의 수를 제한하는 트랜잭션 모니터 역할을 하기도 하고 또는 클라이언트로부터 요청을 받아 여러 서버로 나누어주고 결과를 수집하여 클라이언트에 전해주는 지능적 역할을 할 수도 있다.



(a) 2층 구조 (2 tier-architecture)



(b) 3층 구조 (3 tier-architecture)

(그림 1) 클라이언트/서버 시스템의 구조

3층 구조는 기존의 2층 구조(two-tier)를 대체하고 있다. 데이터베이스 시스템의 경우, 기존의 2층 구조에서는 클라이언트가 데이터 표현(presentation)을 맡고 서버가 데이터베이스를 관리하였다. 2층 구조의 주된 문제점은 필요한 비지니스 로직이 서버나 클라이언트 중 한쪽에만 실현될 수 있다는 것이다. 서버쪽에 실현될 경우 서버는 데이터베이스 탐색 뿐만 아니라 그러한 탐색요청을 조작하는 일까지 해야하기 때문에 과도한 부하가 걸리게 된다. 클라이언트 쪽에 실현이 된 경우, 클라이언트 소프트웨어는 획일적이 되거나 플랫폼에 의존적이 된다. 3층 구조에서는 클라이언트와 서버 사이에 다른 하나의 층을 더하여 데이터베이스와 실제 데이터 표현 사이를 분할하여 준다. 대개의 경우 최상부층에는 가장 강력한 성능의 시스템과 공용 정보원이 되는 중대형 컴퓨터를 놓고 중부층은 강력한 LAN 서버를 배치한다. 이들은 중대형 컴퓨터에게 적절한 요청사항을 보내는 최상부층 클라이언트로써의 역할과 동시에 하부층에 있는 워크스테이션이나 PC들의 서버 역할까지 한다.

일반적으로 2층구조에 비교하여 3층 구조는 다음과 같은 장점을 갖는다. 첫째, 여러 곳에 분산된 데이터를 동시에 실시간으로 접근하는 것이 가능하며 프로세스의 병렬처리를 통하여 사용자와 데이터의 양이 증가해도 상대적으로 일정한 속도를 유지할 수 있다. 둘째, 클라이언트 PC가 데이터를 접근하는 중에 down 되어도 서버 프로세스와는 무관하고 로직이 서버에 존재하여 locking 시간이 짧으므로 deadlock 발생 가능성이 적다. 셋째 로직이 서버에 구현되어 있으므로 서버에서 작업한 결과만 클라이언트로 전송하므로 네트워크의 교통량을 줄일 수 있다. 그러나 설계가 어렵고 시스템이 복잡해짐에 따라 하드웨어와 소프트웨어의 비용이 증가하는 것은 3층 구조를 가진 시스템의 단점이라 할 수 있다.

3. 클라이언트/서버 컴퓨팅 모델

Web을 포함한 인터넷이 어떻게 동작하는지를 이해하기 위해서는 클라이언트/서버 컴퓨팅의 개념을 이해해야 한다. 클라이언트/서버 모델은 한 프로그램(클라이언트)이 정보교환을 위하여 다른 프로그램(서버)과 통신하는 분산 컴퓨팅의 한 가지 형태이다. 클라이언트/서버 모델에서 클라이언트는 주로 하나 또는 그 이상의 서버들에 의해 제공되는 서비스의 사용자 프로그램이다. 이 모델에서는 서버가 클라이언트로부터의 요구에 응답하는 서비스 제공자 역할을 한다는 원칙에 기초하여 명확한 기능의 분할이 이루어진다. 한 머신이 어떤 경우에는 서버의 역할을 하면서 다른 경우에는 클라이언트의 역할을 할 수 있다는 점을 이해하는 것이 중요하다. 예를 들어, LAN 관리자 환경에서는 어떤 머신이 한 사용자를 위한 클라이언트로 사용되면서 동시에 다수의 사용자를 위한 프린터 서버의 역할을 수행할 수 있다.

서버는 응용, 파일, 데이터베이스, 프린트, 팩스, 화상, 통신, 보안, 시스템, 네트워크 관리 등의 서비스를 제공한다. 서버는 물리적 구현을 기술하는 것이 아니라 구조적 개념이다. 클라이언트와

서버 기능은 동일한 물리적 장치를 통해 제공될 수도 있다. 상호 이용 컴퓨팅(peer computing)으로 전환됨에 따라 잠재적으로 모든 장치는 서비스의 요청에 응답하기 위해 클라이언트와 서버의 역할을 동시에 수행할 것이다.

응용 서버는 클라이언트의 사용을 지원하기 위한 업무처리 기능을 제공한다. 클라이언트/서버 모델에서 응용 서비스는 업무 기능의 전체 또는 일부를 처리하기 위해 제공될 수 있으며 프로세스간 통신을 통한 서비스 요청으로 호출된다. 이를 위하여 메시지를 통한 요청이나 원격 프로시져 호출(RPC)이 이용될 수 있다. 하나의 전체 업무 기능을 제공하기 위해 일련의 응용 서버들을 같이 사용할 수 있다. 예를 들어, 급여 시스템에서 한 응용 서버는 직원 정보를 관리하고 다른 응용 서버는 소득을 계산하며, 또 다른 응용 서버는 공제액을 계산할 수 있다. 이들 서버들은 여러 하드웨어 플랫폼에서 서로 다른 운영 체제를 사용하고 다른 데이터베이스 서버를 사용할 수도 있다. 일반적으로 클라이언트와 서버의 역할을 다음과 같이 나열할 수 있다.

클라이언트의 역할 :

1. 사용자 인터페이스 처리
2. 사용자의 요구를 필요한 프로토콜로 번역
3. 서버에게 사용자 요구 전송
4. 서버의 응답 대기
5. 서버의 응답을 "human-readable" 결과로 번역
6. 결과를 사용자에게 제공

서버의 역할 :

1. 클라이언트의 쿼리 대기
2. 클라이언트의 쿼리 수행
3. 결과를 클라이언트에게 전송

전형적인 클라이언트/서버 컴퓨팅 과정은 다음과 같다.

1. 사용자가 쿼리를 생성하기 위하여 클라이언트 소프트웨어를 실행
2. 클라이언트가 서버를 연결한다.
3. 클라이언트가 서버에게 쿼리를 보낸다.
4. 서버가 쿼리를 분석한다.
5. 서버가 쿼리를 수행하여 결과를 산출한다.
6. 서버가 결과를 클라이언트에게 보낸다.
7. 클라이언트가 이 결과를 사용자에게 제공한다.
8. 필요에 따라 위의 과정을 반복한다.

이와 같은 클라이언트/서버 컴퓨팅은 중국 음식점에서 행해지는 일에 비유하여 설명할 수 있다. 음식점에서 점원은 손님에게 메뉴를 제공한다. 손님이 주문할 것을 결정한 후에 점원은 주문을 받고 이를 중국어로 번역하여 중국인 주방장에게 전달한다. 주방장이 음식을 준비한 후 점원은 이를 손님에게 제공한다. 정상적이라면 손님이 선택한 음식이 나오겠지만 때에 따라서는 "번역" 상의 문제로 다른 음식이 나올 수도 있다. 이 비유에서 음식점에 간 손님을 클라이언트로 그리고 중국인 주방장을 서버로 생각할 수 있다.

클라이언트/서버 컴퓨팅의 가장 명확한 장점은 융통성 있는 사용자 인터페이스이다. 데이터를 소유하고 있는 서버와는 독립적으로 사용자 인터페이스를 생성하는 것이 가능하다. 그러므로 클라이언트/서버 응용 프로그램의 인터페이스는 매킨토시상에서, 서버는 메인프레임 상에서 그리고 클라이언트들은 도스나 유닉스 컴퓨터 상에서 작성될 수 있다. 이러한 기법은 정보가 중앙의 서버에 저장되어 있고 다양한 형태의 원격지 컴퓨터들로 분산될 수 있음을 가능케 한다. 사용자 인터페이스의 관리는 클라이언트의

역할이므로 서버는 쿼리를 분석하고 정보를 분산하기 위한 작업에 보다 많은 연산 자원을 집중하게 된다. 이것이 클라이언트/서버 컴퓨팅의 또 다른 장점이며 이를 통하여 분산 컴퓨팅 기반을 보다 강력하게 한다. 메인프레임을 서버로 사용하는 것에 비해 매킨토시를 서버로 사용하는 것이 연산이나 저장 능력에 있어서 매우 작지만 불가능한 일은 아니다.

간략히 설명하여 클라이언트/서버 컴퓨팅은 완전히 독립적인 머신들이 하나의 작업을 수행할 수 있도록 하는 메카니즘이다.

4. 다중 서버를 위한 Coordination 모델

기본적인 클라이언트/서버 모델은 클라이언트가 단일 서비스를 요구하는 응용 프로그램을 대상으로 한다. 그러나 경우에 따라서 클라이언트는 다수개의 서버로부터의 서로 다른 서비스를 요구하고 이들이 서로 연관성 있게 수행됨으로써 클라이언트가 필요로 하는 응용 프로그램을 수행한다. 이와 같은 연산 모델은 workgroup system의 태스크 할당이나 workflow application에서의 태스크 순차화 등의 예에서 발견된다. 이러한 연산 모델에서는 시스템을 구성하는 요소들, 즉 클라이언트와 다수개의 서버간의 상호작용을 위한 제어가 매우 복잡하므로 단일화된 연산 도구의 개발이 지체되어 왔다. 본 절에서는 다중 서버 클라이언트/서버 환경을 모델링 하기 위한 고려사항과 기술들을 설명한다.

다중 서버 클라이언트/서버 모델의 설계를 위하여 고려해야 하는 기본적인 문제들은 다음과 같다.

- 다중 서비스가 어떻게 요청되는가 ?
- 이 서비스들이 어떻게 관리되는가 ?
- 클라이언트들이 어떻게 응답을 얻는가 ?

다중 서비스를 요구하는 클라이언트의 응용 프로그램을 설계할 때, 클라이언트의 위치와 그것이 여러개의 서버들과 연동하는 방법이 중요하다. 소규모의 정적 시스템에서는 클라이언트가 서버들이 지원 가능한 서비스가 무엇인지를 판단하고 각 서비스를 지원하는 서버를 구분하여 서버에 접근하는 작업이 비교적 쉬운 일이다. 그러나 시스템의 규모가 확장되고 서버들의 구성이 동적으로 추가, 삭제, 재구성에 의하여 변화하는 경우에는 이 작업이 매우 복잡해진다. 특히 이기종 시스템의 경우에는 서버에 따라 클라이언트와의 인터페이스가 다양하므로 서버로의 접근 문제가 더욱 복잡하다. 따라서 클라이언트 프로그램 개발자는 서버로의 접근 문제를 해결하기 위한 시스템 수준의 지원을 필요로 하며 이를 중간서버(intermediate server)라 한다. 클라이언트/서버 모델의 맥락에서 보면 중간서버도 하나의 서버 즉, 자원서버(resource server)로 취급될 수 있다. 이러한 중간서버를 포함하는 클라이언트/서버 모델에서 클라이언트는 서버에 직접적으로 접근하거나 중간서버를 통하여 간접적으로 접근할 수 있다.

클라이언트는 다중 서비스를 explicit 하게 혹은 implicit 하게 요청할 수 있다. Explicit 요청 방법에서는 클라이언트가 필요한 각각의 서비스에 대하여 하나의 요청을 보낸다. 이 때, 클라이언트는 각 서버가 제공하는 서비스들에 대한 정보를 관리하고 이를 종합하여 서비스를 요청하기 위한 서버를 결정해야 하므로 클라이언트의 작업 부하가 크다. Implicit 요청에서는 클라이언트가 complex 서버에게 하나의 종합적인 요청을 보내고 이 서버가 클라이언트의 요청을 만족시키기 위하여 다른 서버들과 접촉하게 된다.

이기종 네트워크상에 분산된 데이터베이스에서의 정보검색의 경우, 클라이언트에 의하여 제기된 정보 검색 명령의 분할, 세부검색 명령을 위한

서버의 선정, 각 서버로부터의 수행결과 종합 등의 작업이 필요하다. Implicit 요청 방법에서는 complex 서버가 이러한 일련의 작업을 담당하므로 클라이언트의 입장에서는 투명성 있는 서비스가 제공된다.

중간서버에 의하여 관리되는 서비스들은 상호 독립적인 서비스로 구성되거나 의존관계를 지닐 수 있다. 일반적으로 서비스들간의 의존관계는 자료의존성이나 동기화 문제 등을 의미한다. 의존관계에 있는 서비스들은 어떤 서비스에 의해 산출된 결과가 다른 서비스의 입력으로 제공되거나 분기문을 결정짓는 조건으로 사용될 수 있다. 어떤 서비스는 다른 서비스에 의한 상태변화를 수용해야 하기 때문에 자신의 수행을 위하여 다른 서비스의 수행이 완료되기까지 대기해야 하는 경우도 있다. 서비스간에 이러한 의존관계가 존재할 때, 이를 해결하는 것은 중간서버의 역할에 포함된다. 이와는 달리, 상호 독립적인 서비스들의 경우에는 서비스들의 실행 순서가 고려되지 않으므로 클라이언트와 서버들을 연결하는 중간서버의 역할이 비교적 간단히 구현될 수 있다.

4.1 Service Request Manager (SRM)

Broker 모델은 클라이언트 응용 프로그램과 클라이언트가 요구하는 서비스를 제공하는 서버들을 중재하는 가장 간단한 형태의 제어 기법이다. Broker는 클라이언트로부터 원하는 서비스를 받기 위하여 어떤 서버를 선정해야 하며 어떤 방법으로 서버에 접근할 것인지에 대한 정보를 관리해야 하는 부담을 덜어 줄 수 있다. 분산환경에 속하는 모든 응용 프로그램들은 그들이 지원하는 서비스, 응용 프로그램의 위치, 클라이언트와의 인터페이스 등을 broker에 등록한다. Broker는 이러한 서비스 정보들을 naming service나 디렉토리 형태로 관리한다. Broker에 따라 서비스 정보는 컴파일시에 정적으로 등록되거나 실행시간에 동적으로 등록된다. Broker는 다음과 같이 3가지 종류가 있다.

1) Forwarding broker

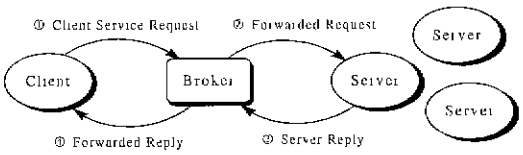
이 모델에서는 broker가 클라이언트의 서비스 요구를 적절한 서버에 전달하고, 서버의 응답을 얻어 이를 클라이언트로 되돌리는 역할을 한다. 클라이언트는 원하는 서비스를 broker에게 요구한다. Broker는 관리되고 있는 서비스 정보들을 이용하여 요구된 서비스를 제공하는 서버를 찾아 클라이언트로부터 전달된 서비스 요구를 전달한다. 서버 또한 계산된 결과를 broker에게 전달하며 broker는 전달받은 결과를 클라이언트로 되돌린다.

2) Handle-driven broker

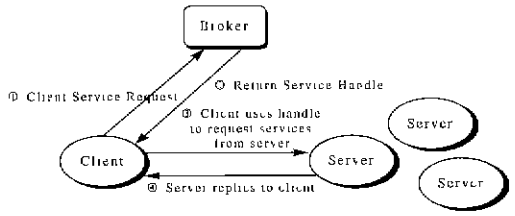
이 모델에서 broker의 역할은 클라이언트의 서비스 요구에 대하여 클라이언트가 원하는 정보의 집합인 "handle"을 클라이언트로 전달하는 것이다. "Handle"은 클라이언트가 원하는 서비스를 제공하는 서버의 이름, 네트워크상에서 서버의 노드 주소, 서버로의 접근을 위한 형식 등을 포함한다. Broker로부터 handle을 제공받은 클라이언트는 이 정보에 기초하여 서버로 직접 접근하며 서비스를 수행한 서버는 실행 결과를 클라이언트로 직접 되돌린다.

3) Hybrid broker

이 모델은 Forwarding broker와 Handle-driven broker의 혼합된 형태로서 클라이언트가 broker에게 명령을 전송할 때, 어떤 모델의 broker를 사용할 것인가를 명시하는 형태이다.



(a) Forwading broker



(b) Handle-driven broker

(그림 2) Broker 모델

위의 세 가지 모델 중 forwarding broker 모델은 모든 클라이언트와 서버들간의 서비스 요청을 처리하는 작업이 중앙의 broker에서 일괄적으로 처리되므로 오류 검색이나 복구 등의 문제에 대하여 클라이언트가 관여하지 않아도 되므로 클라이언트의 역할이 다소 간단하다. 그러나 broker로의 과도한 메시지 전송이 전체 성능을 감소시킬 우려가 있다. 반면 handle-driven broker는 broker의 기능이 매우 간단하고 결과가 broker를 거치지 않으므로 병목현상에 대한 우려가 적다. 또한 클라이언트가 동일한 서비스를 반복적으로 요구할 경우에는 broker로부터 제공받은 handle을 클라이언트에 저장하여 재 사용할 수 있으므로 broker의 통신량을 줄이는 효과를 얻을 수 있다. 그러나 이 방법은 오류에 대한 처리를 클라이언트에게 부과하므로 클라이언트 프로그램이 복잡해 질 수 있다. Hybrid broker 모델은 두 가지 모델의 장점을 이용한 것으로 broker 이용의 융통성을 높일 수 있으며 각 클라이언트의 응용 프로그램 및 환경을 고려하여 적절한 제어를 선택적으로 사용할 수 있다.

4.2 Process Planner

Implicit한 서비스 요청을 하는 클라이언트는 하나 혹은 그 이상의 complex 서버를 통하여 자원서버에 간접적으로 접근한다. Complex 서버는 클라이언트의 서비스 요구를 받아 이를 각 서버가 제공하는 세부 서비스로 재구성한 후 각 서비스를 제공하는 서버들에게 명령을 전달한다. 각 서버들은 서비스의 실행을 완료하고 그 결과를 complex 서버에게 보내며 complex 서버는 이들을 종합하여 클라이언트에게 전달한다. 이 때, 세부적으로 재구성된 서비스들이 서로 독립적이고 비동기적 통신 기법을 사용하는 경우에는 complex 서버가 서버들에게 병렬적인 서비스 수행을 요구할 수 있다. 이러한 방식은 다중 프로세서 시스템에서 프로세스들의 fork-join 명령이 하나의 연산을 독립적인 thread들로 나누어 동시에 수행하고 결과를 병합하는 것과 유사하다.

반면, 재구성된 서비스간에 의존성이 존재하는 경우에는 이들의 수행 순서를 결정하거나 동기를 이룰 필요가 있다. 이런 경우, complex 서버는 서비스들간의 관계를 분석하여 서비스의 요청 및 결과를 얻기 위한 순서를 관리해야 한다. 이러한 complex 서버를 process planner라 하며 다음 알고리즘에 기술된 기능을 수행한다.

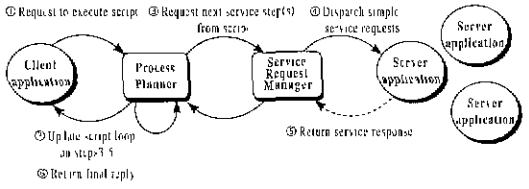
(1) 클라이언트가 요청한 서비스를 적절한 세부 서비스들로 재구성한다.
(2) 각 세부 서비스들을 수행하기 위한 서버를 선정한다.
(3) 서비스들간의 의존관계를 결정하여 의존관계 목록을 생성한다.
(4) 모든 세부 서비스들이 완료될 때까지 다음 과정을 반복한다.
  (a) 의존관계가 부합되는 서비스들을 각 서버에 할당한다.

(b) 실행을 끝낸 서버들로부터 응답을 받는다.
(c) 의존관계 목록을 갱신한다.

(5) 각 서버들로부터 얻어진 결과를 최종적으로 종합한다.

(6) 클라이언트에게 결과를 되돌린다.

이 알고리즘에서 process planner는 broker의 기능을 함축하고 있다(2, 4a, 6). 이 기능들 이외의 단계, 즉 클라이언트의 서비스를 재구성하고 의존관계 목록을 생성하며 서버들로부터의 결과를 종합하는 등의 작업은 command procedure 혹은 script를 이용한다. Control engine은 이러한 script를 이용하여 부분 서비스를 수행하고 서비스들간의 관계를 갱신, 관리한다. 이러한 프로세스-oriented 모델은 서비스 관리를 위한 정보나 태스크들의 자동 라우팅을 지원하며 explicit 프로세스 모델과 부합됨을 보장한다. 이러한 script 이용의 중용한 응용 예로써는 사무자동화나 프로세스제어, concurrent engineering system 등을 포함한다.



(그림 3) Processor planner

(그림 3)은 process planner의 기능적 모델을 설명한다.

① 클라이언트는 클라이언트 API(Application Programming Interface) 명령을 부름으로써 이미 정의되어 있는 script를 실행시킨다. 이때 API 명령은 process planner에게 메시지를 전달하며 이 과정은 클라이언트 사용자에게는 투명성 있게 실행된다.

② Process planner는 script 번역기를 내재하고 있어서 명시된 script를 번역하고 script를 단계별로 실행한다. script의 실행은 임시변수의 값을 갱신하며 필요에 따라서는 의존성을 검사하기도 한다. script에 명시된 각 서비스 요구의 실행은 단계 ③-⑤를 반복하며 실행된다.

③ Script 번역기는 script의 각 단계를 수행하기 위하여 다른 머신에 있는 서비스 관리자(SRM)에게 서비스 요구를 전송한다.

④ 각 머신에 있는 SRM은 script 번역기로부터 전송받은 서비스 요구를 이에 해당하는 서버와 연결한다.

⑤ SRM은 서버로부터의 응답을 받아 process planner에게 전달한다. 결과를 전달받은 process planner는 script를 갱신하고 의존관계가 해결된 다른 과정의 script에 대한 서비스를 요구한다.

⑥ 모든 script의 실행이 끝난 시점에서 process planner는 결과를 클라이언트에게 전달한다.

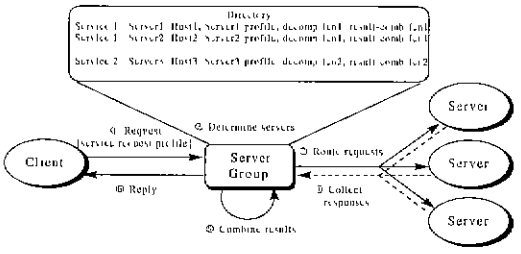
4.3 Server Group

Server group 모델은 클라이언트가 상호 독립적인 여러가지 서비스를 요청할 경우 이를 수행할 수 있도록 해 준다. Process planner가 SRM (Broker)의 기능을 이용하여 구현되는 것과는 달리, server group은 독립적인 제어 서버로서 동작한다. 특히 server group은 서비스 등록과 서비스 수행을 위하여 SRM API를 확장한 SRM의 하위 부류이다.

server group은 SRM의 기본기능인 각각의 서비스에 대한 등록 및 요청 뿐 아니라 복합 서비스를 서버 그룹 디렉토리에 등록할 수 있게 해 준다. 디렉토리에는 복합 서비스를 각각의 서비스로 나누고 결과를 종합하는 함수의 포인터가 저장되어 있다. 클라이언트는 server group을 하나의 서

버로 간주하여 복합 서비스를 요청할 수 있다. 클라이언트의 요청에 대한 server group의 역할은 복합 서비스를 각 서버가 지원하는 각각의 서비스들로 분할하여 할당하는 것이다. 특히 server group에서의 복합 서비스 분할은 각 서비스들이 독립적으로 수행될 수 있도록 되어야 하며 따라서 클라이언트가 요구한 서비스는 작은 단위로 여러 서버에서 동시에 수행된다. 서비스의 최종 결과는 각 서버들의 실행결과를 종합하므로써 얻을 수 있으며 서비스 응답을 받는 클라이언트는 server group과 서버들간의 상호작용과는 전혀 무관하다. 실행 결과를 종합하기 위하여 사용되는 함수들로는 합집합, 교집합, relational-join, voting 알고리즘과 같은 것들이 있다.

예를 들어, 클라이언트가 복잡한 시스템의 오류를 검색하는 서비스를 요청할 경우, 이 서비스는 시스템의 각 모듈에 대한 오류 검색 서비스들로 분할된다. 각 모듈의 오류 검색 결과는 논리합 연산을 통하여 시스템 진단 결과 요약문으로 종합될 것이다.



(그림 4) Server Group

Server group의 두 번째 역할은 어떤 서비스를 제공하는 서버가 다수개 존재할 경우, 이들 중 서비스를 제공할 서버를 선정하는 것이다. 일반적인 Broker 모델에서는 등록된 각 서비스를 제공하는 서버가 시스템 내에 하나라고 가정한다. 이 가정은 서비스에 대한 서버의 선정을 명확하게 한다. 그러나 경우에 따라서는 단일 서비스를 위한 다중 서버의 사용이 자원의 가용성을 높이고 여러 개의 문제 해결 가능성을 강화하므로 바람직할 수 있다. 예를 들어, 시스템 오류의 검색(isolation)은 오류 트리, 신경망, 전문가 시스템 등의 여러 가지 방법에 의하여 해결될 수 있다.

Server group은 다중 서버에서 하나의 서버를 선택하기 위하여 모델링과 제어 구조를 포함한다. 첫째, SRM registration API를 확장하여 하나의 서비스를 위하여 다수개의 서버가 server group에 등록하도록 한다. 같은 서비스를 지원하는 각 서버는 서버들의 상대적인 정확도, 완성도, 태스크 수행 속도 등의 정보를 포함하는 server profile을 이용하여 구분할 수 있다. 예를 들어, 오류 검색을 위하여 신경망 서버와 전문가 시스템이 서버로 이용되는 경우 이들은 실행속도에 있어서 차이를 보인다.

둘째, server group은 SRM 요청 API를 확장하여 클라이언트가 서비스를 요청할 때 자신이 원하는 서버의 범위를 지정하거나 server profile의 속성값을 직접 지정할 수 있도록 한다.

셋째, server group이 기본 SRM 제어 동작을 수정하여 클라이언트의 요청을 수행하기에 적합한 서버를 선정하도록 한다. 또한 클라이언트가 제시한 서버의 조건에 기초하여 서버를 찾아내는 알고리즘이 server group에 추가되어야 한다. server group은 조건에 합당한 서버들에게만 서비스를 요청하고 이들로부터의 결과를 서비스 요청과 함께 명시된 조합 함수(combination function, union, intersection 같은 것)에 의하여 조합한다.

(그림 4)는 server group의 연산 모델을 보여준다. 클라이언트의 서비스 요청에 따라 server group은 복합 서비스를 분할하고 서비스 요구에 명시된 조건에 기초하여 매핑 알고리즘을 적용하므로써 적합한 서버를 결정한다. 서버가 결정되면 서비스를 요구하고 결과들을 수집하여 본래의 서

비스에 해당하는 결과로 조합하여 이 결과를 클라이언트에게 전달한다.

5. 클라이언트/서버 소프트웨어의 경향

근래의 소프트웨어 산업은 분산 컴퓨팅에 의하여 주도되어 왔으며 최근에는 클라이언트/서버 컴퓨팅이 각광 받고 있다. 분산 컴퓨팅의 궁극적 목적은 완전 분산형 상호 이용 컴퓨팅(fully distributed peer-to-peer computing)을 구현하는 것이며 클라이언트/서버 컴퓨팅은 이 목표로 향하는 과도기적인 단계이다. 객체지향 기법, document-centric 소프트웨어 구조, data warehouse 기술, 규격화, end-user programming 경향 등이 완전 분산형 상호 이용 컴퓨팅을 향한 발전을 지원하는 중요한 기술들이다. 여기서 규격화란 분산 컴퓨팅 구조의 발전을 증진시키는 것을 의미한다. 분산컴퓨팅은 server-centric, client-centric, peer-peer 컴퓨팅의 형태로 발전해 왔으며 완전 분산형 상호 이용 컴퓨팅을 지향하고 있다.

Middleware는 분산컴퓨터 네트워크에서 노드들을 묶어주는 역할을 하는 소프트웨어로서 Fully distributed peer-to-peer computing(완전 분산형 상호 이용 컴퓨팅)을 가능하게 하는 핵심적인 기술중의 하나이다. Middleware의 특성 및 기능에 관한 명확한 규정은 아직 결정된 바 없으나 일반적으로 다음과 같은 서비스를 사용자에게 투명적으로 제공하여야 한다.

1) Communication service

IPX/SPX와 TCP/IP를 포함한 많은 통신 프로토콜들이 표준으로 제정되었다. 분산 환경은 다양한 통신 프로토콜을 사용하는 컴퓨터들로 구성될 수 있으므로 이들간의 통신을 위해서는 middleware가 현재 사용중인 프로토콜과 앞으로 사용될 다양한 프로토콜을 지원할 수 있어야 한다. 뿐만 아니라, 하나의 프로토콜이더라도 서로 다른 벤더에 의하여 구현된 프로토콜들은 차이점을 지닐 수 있다. 예를 들어 TCP/IP의 경우, 현재 사용되고 있는 제품이 15가지이며 Window 95에 의해서도 새로운 제품이 선보일 예정이다. 따라서 middle- ware는 한 표준에 대하여 서로 다른 제품의 차이를 수용할 수 있어야 한다. 또한 네트워크 환경내에서 다른 프로토콜을 사용하는 머신간의 통신을 위하여 프로토콜 번역(해석) 기능이 필요하다.

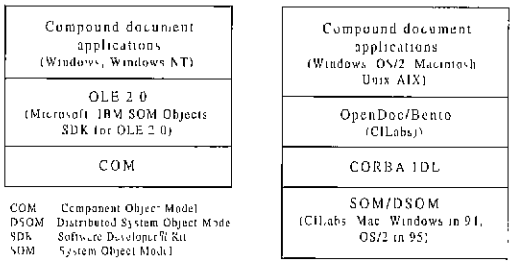
2) Data Access and Connectivity Services

클라이언트와 서버를 연결하기 위한 표준 API들이 많이 나와 있다. 이들은 SQL oriented이거나 범용이다. 뿐만 아니라 OLE나 DSOM과 같은 객체형 표준들도 사용되고 있다. 따라서 middleware는 이러한 표준들을 제공하는 기능을 포함해야 한다. 클라이언트를 위한 데이터가 분산된 서버들에 나뉘어져 있을 경우에 middleware는 이들을 탐색하는 기능을 포함해야 한다. 그리고 서로 다른 파일 구조나 인덱싱 기법들을 필요로 하는 서비스 요청에 대한 대처방안도 고려되어야 한다.

3) Scheduling services

Thread 관리는 프로세스간의 통신을 가능하게 하고 CICS나 IMS/DC와 같이 트랜잭션에 기초한 환경에서의 보안 유지를 가능하게 한다. 이러한 환경들은 다수개의 프로세스를 동시에 관리할 수 있도록 한다. 서로 다른 머신 환경에서는 이와 같은 기능들이 다르게 취급되므로 middleware는 환경의 변화에 따라 응용 프로그램이 잘 적응하도록 하는 기능을 수행해야 한다. 하나의 시스템 자원에 여러 사용자가 접근하고자 할 때, 사용자의 입장에서는 이러한 충돌을 고려하지 않으므로 이를 해결하는 것 또한 middleware의 역할이다. Middleware는 머신간의 작업 부하 유지 기능을 포함해야 하고 자원을 사용하는 응용 프로그램들의 우선순위를 고려하여 고성능을 요하는 응용

프로그램들이 우선적으로 자원을 사용할 수 있도록 하는 기능을 필요로 한다.

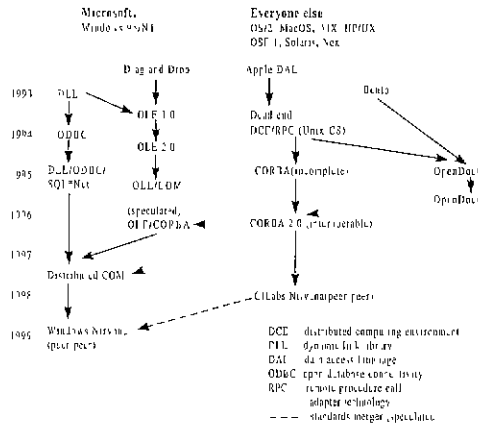


(그림 5) Middleware 소프트웨어 계층

미래의 분산환경에서 middleware는 PC 수준의 컴퓨터로부터 대규모 컴퓨터에서 사용될 응용 프로그램의 infrastructure가 될 것이다. 따라서 이것이 미래의 소프트웨어 시장의 핵심이 될 것이므로 많은 기업체가 middleware 개발에 주력하고 있다. 현재 소프트웨어 시장을 이끌고 있는 두 가지 middleware 구조는 Mircrosoft사의 OLE/COM (Component Object Model)과 Component Integration Laboratories의 OpenDoc/DSOM이다. 이들 두 가지 구조는 공통적으로 document-centric 컴퓨팅을 지원한다. (그림 5)는 각 middleware의 계층구조를 보여준다.

OLE와 OpenDoc는 각각 COM(Component Object Model) 계층과 SOM(System Object Model) 계층위에 구현되었다. (그림 5)에서 OpenDoc의 Bento는 문서들간의 파일 저장 형식이며 CORBA(Common Object Request Architecture) IDL은 SOM/DSOM 서비스간의 인터페이스를 정의하기 위한 언어이다.

Middleware 산업에 있어 이들 양대 산맥의 존속 여부는 컴퓨팅 산업의 중요한 관심사가 되고 있다. Middleware 산업에 있어서 각 기업의 승패 여부는 이것이 개방형 소프트웨어가 될 것인가 혹은 toll road가 될 것인가 하는 것이다. 특히, 현재 전세계적인 데스크탑의 보급율을 간과할 수 없으므로 데스크탑 상에서 수행되는 middleware에 대한 우위를 차지하는 기업이 전체 산업 경쟁에서 우위를 차지할 것으로 예상할 수 있다. 만일 한 기업의 middleware가 전체 infrastructure를 차지하게 된다면 이를 응용한 소프트웨어들은 그 기업의 middleware를 바탕으로 구현되어야 할 것이다. (그림 6)은 예상 가능한 middleware 산업의 발전 경로를 제시하고 있다.



(그림 6) Middleware 산업의 발전 방향

6. 결 론

본 고에서는 분산 환경에서 사용되는 클라이언트/서버 컴퓨팅 기법에 관하여 고찰하였다. 클라이언트/서버 환경은 한정된 자원 및 데이터의 공유 뿐 아니라 응용 프로그램의 효과적인 처리를 위하여 관심이 집중되고 있다. 초기의 클라이언트/서버 환경은 클라이언트와 서버가 네트워크를 통하여 직접 연결되고 상호작용에 대한 기능도 직접 담당하는 형태의 2층 구조로 설계되었다. 그러나 분산 환경을 구성하는 클라이언트 및 서버의 수가 많아지고 제공되는 서비스가 다양해짐에 따라 클라이언트와 서버 사이에 하나의 층을 더하여 상호작용을 수행하도록 하는 3층 구조가 널리

사용되고 있다. 뿐 만 아니라 클라이언트/서버 컴퓨팅을 이용하여 복잡한 서비스를 제공 받고자 하는 응용 문제를 위해서는 서버들간의 수행 및 서버와 클라이언트의 상호작용을 관리하기 위한 컴퓨팅 모델이 더욱 복잡해 진다. 이러한 문제들에 대하여 클라이언트와 서버 사이에 중간서버(intermediate server)를 구축하고 서비스의 번역, 서버의 호출, 결과의 수집, 종합 등의 기능을 부여함으로써 다중 서버 클라이언트/서버 컴퓨팅을 효과적으로 구현할 수 있다. 이와 같은 컴퓨팅 모델에 기초하여 네트워크상의 분산된 컴퓨터들을 연결시켜주는 클라이언트/서버 소프트웨어인 middleware가 미래의 소프트웨어 시장에서 주요한 부분을 점유할 것으로 예상된다. Middleware는 클라이언트/서버 컴퓨팅이 궁극적으로 지향하는 완전 분산형 상호 이용 컴퓨팅(fully distributed peer-to-peer computing)을 구현하기 위한 핵심적인 기술이며 middleware 산업이 미래 소프트웨어 산업을 선도하게 될 것이다.

참고문헌

[1] P. Smith and S. Guengerich, Client-Server Computing, Sams, 1992.

[2] A. Berson, Client-Server Architecture, McGraw-Hill, 1995.

[3] R. M. Adler, "Distributed coordination models for client/server computing," *COMPUTER*, Vol. 28, No. 4, pp.14-22, April, 1995.

[4] T. G. Lewis, "Where is client/server software headed?," *COMPUTER*, Vol. 28, No. 4, pp.49-55, April, 1995.

[5] J. M. Andreoli and R. Pareschi, "Integrated computational paradigms for flexible client-server communication," *ACM Computing Serveys*, Vol. 28, No. 2, June, 1996.

[6] P. J. Douglas, G. M. Alliger, and R. Goldberg, "Refining the curriculumn: Client-server and object-oriented training," *COMPUTER*, Vol. 29, No. 6, pp.80-84, June, 1996.

문 현 주

1995년 충북대학교 컴퓨터과학과 졸업 (이학사)
1997년 충북대학교 대학원 전자계산학과 졸업 (이학석사)
1997년-현재 충북대학교 대학원 전자계산학과 박사과정
관심분야 : 분산처리, 슈퍼컴퓨팅, 병렬처리

황 인 재

1986년 충북대학교 컴퓨터공학과 졸업 (공학사)
1991년 University of Florida, Computer & Information Sciences 졸업 (공학석사)
1994년 University of Florida, Computer & Information Sciences 졸업 (공학박사)
1986년-1987년 한국 전자통신연구소 연구원
1995년-현재 충북대학교 컴퓨터교육과 조교수
관심분야 : 병렬처리, 병렬컴퓨터구조, 병렬 알고리즘

김 석 일

1975년 서울대학교 전기공학과 졸업 (공학사)
1975년-1990년 국방과학연구소 선임연구원
1985년-1989년 North Carolina State University 졸업 (공학박사)
1990년-현재 충북대학교 컴퓨터과학과 부교수
관심분야 : 병렬처리 컴퓨터구조, 분산처리, 슈퍼컴퓨팅, 병렬처리 언어