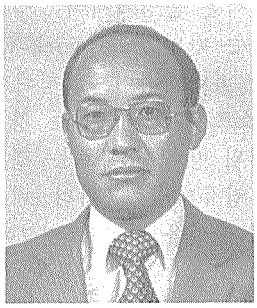
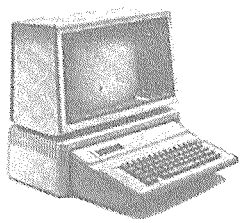


吳 吉 祿
韓国電子通信研究所
컴퓨터연구부장 / 工博

분산처리의 운영체제



*본고는 컴퓨터 연구부내의 Technical Memo로 박기완, 손덕주 선임 연구원이 작성한 것임.

1. 개요

Distributed Computer System은 통신 subnet와 S/W에 의해 logical하게 연결된 processor와 memory pair의 collection이라 할 수 있다.

이러한 분산 시스템이 점점 중요해지고 관심이 높아지는 이유는 근래 Microelectronics 의 급속한 발달로 우수한 Component가 싼값에 available하고, CPU processing, storage와 data 전송의 단가가 상당히 떨어졌음을 보았으며, 사용자의 요구 또한 변화하여서 쉽게 사용할 수 있는 시스템과 Service가 expandable하고, 사용자들의 요구에 맞도록 쉽게 변화가 가능해야 하고 보통의 조직은 중앙 집중된 형태이지만 조직에 따라서는 작업의 분권화가 필요하고 local task들은 그 업무를 가장 잘 이해하는 사람들에 의해서 수행되고 제어되면 더욱 능률적이 될 수도 있으며 Computing power가 필요한 곳에 processing element를 설치함으로써 통신비를 절감할 수도 있기 때문이다.

일반적으로 분산처리 시스템에서 기대할 수 있는 것은 다음과 같다.

첫째, 성능이 향상되도록 기존의 multiprocessor에 존재하는 bottleneck 즉 shared memory based interprocessor communication 및 control process scheduler 등이 제거되어서 overall system control을 달성하도록 한 작업에 협동하는 여러 processing element들을 decentralized technique으로 상호연결하여, 성능을 향상하여야 한다.

둘째, 확장 가능한 시스템은 그의 기능을 손상하지 않고 성능 요구의 변화 및 기능 요구의 변화와 같은 환경의 변화에 쉽게 적응할 수 있는 시스템이다. Modular architecture는 분산처리 시스템의 한 특성으로써 시스템 설계의 간

단함, 설치의 용이, 유지 보수의 용이와 같은 장점이 있다.

세째, critical resource 하나에 의존하는 시스템의 availability는 매우 낮으므로 processing element가 여러개 있고 이들이 서로를 감시하도록 하면 하드웨어, 소프트웨어, 데이터의 redundancy의 잇점을 얻을 수 있으므로 fail-soft computing system을 달성할 수 있다.

마지막으로 physical device, data base, processing element와 같은 자원 공유는 load sharing 및 실제 구조에 대해서 transparency 가 있을 것을 요구한다. 여러 자원의 remote access도 가능해야 하며 최적이고 동적인 자원할당과 자원 공유를 달성할 목적으로 시스템의 모든 행동에 대해서 system-wide control이 요구된다.

2. Design Approach

분산 시스템은 여러 형태로 만들어 질 수 있으나 여기서는 Local Area Network에 의해 연결된 Computer들로 가정하며, 분산 시스템의 application이 가능하도록 하는 software는 기계 사이에 자료전송을 위한 explicit한 mechanism이 있는 경우와, 어떤 resource를 identify하는 방법과 기계사이의 inter-processor communication을 제공하여 어느 때라도 resource의 reference가 가능하도록 한 경우 그리고, 완전히 abstract한 resource naming scheme 을 implement함으로써 지시하는 object의 실제의 location을 몰라도 되도록 하는 경우가 있을 수 있다. 그들의 차이는 naming scheme을 실현하는 방법과, 그 방법을 어느 level에서 시스템에 integrate되어 있는가에 있다.

컴퓨터 네트워크 환경에서 각 사용자가 각 machine마다 account를 갖고 각 machine의 명령어를 배우고 또 프로그램과 데이터의 분산 관리를 각 사용자가 알아서 한다는 것은 바람직하지 않다. 그러므로 자원과 computing을 일관성있게 관리하는 network-wide 운영체제가 필요하게 된다. 그러한 시스템은 몇가지 방법으로 건조될 수 있다.

네트워크 운영체제(Network Operating System)라 불리는 것은 각 host가 기존운영 체제를 그대로 유지하면서 네트워크 운영체제를 그 위에 구현하는 것이다. 이 방법은 구현하기 쉽고 기존의 소프트웨어를 계속 사용 가능하다. 또 다른 방법은 기존의 운영체제를 쓰기보다는 단일 동종의 분산 운영 체제(Distributed Operating System)를 사용하는 것이다. 장기적으로 볼 때는 호환성이 적은 운영체제의 집합을 얼기설기 엮는 것보다는 단일의 network-wide 운영체제를 갖는 것이 바람직하다. 대형의 host들과 중요한 기존의 소프트웨어가 있는 경우에는 네트워크 운영체제의 방법을 따르는 경향이 있고 중형 또는 소형 컴퓨터의 근거리 네트워크는 분산운영 체제의 방법을 따르는 경향이 있다.

네트워크 운영체제는 기존의 운영 체제들을 단일 연합 체제로 변환시키기에 적합하다. 이것은 기존의 소프트웨어 위에 분산층(distribution layer)을 구현함으로써 가능하다. 이 분산층에서는 서비스가 요구될 때 parameter로서 전해지는 object의 name을 해석하고 그것이 local system에 있는지 remote에 있는지를 결정한다. 만약 object가 local이면 요구되는 작업은 native O. S. 에 의해서 수행되고 remote인 경우는 그 작업을 수행하고 결과를 보내줄 것을 remote system에 요구하게 된다.

Distributed Operating Systdm 의 방법으로는 LOCUS system과 같이 scratch부터 새로

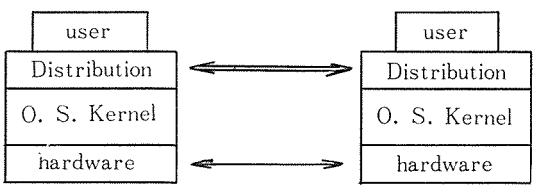


그림 1 NOS 모델

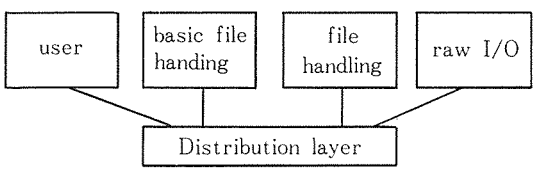


그림 2 Functional Distribution 모델

이 design된 경우이며 개발비가 높은 반면, 처음부터 효율적인 설계가 가능하여 높은 performance를 기대할 수 있다.

Distribution layer를 구현한 다음 그위에 system interface를 위한 software를 implement하는 방법으로 Functional Distribution이 있으며 network wide한 효율적인 inter-process communication을 실현하는 것이 중요하다. Accent kernel이 Functional Distribution model로 사용될 수 있으며, Cambridge Distributed System이 이런 성격을 띠고 있으며, 확장성이 뛰어난 장점이 있다.

3. Issues in Distributed Computer Systems

Distributed Computer System은 Communication Subnet, Local Area Network, Distributed Operating System, Distributed databases, Concurrent and Distributed Programming Languages, Specification Languages for concurrent systems, Theory of parallel algorithms, Parallel Architectures and Interconnection Structures, Fault Tolerant and unreliable Systems, Distributed Real-time Systems, Cooperative problem solving techniques of artificial intelligence, Distributed debugging, Distributed simulation, 그리고 Distributed application을 포함하는 광범한 research area이다. 여기서는 Distributed Operating System만 주로 언급하려고 한다.

이러한 광범위한 area에서 공통되는 Issue들은, The Object Model, Access Control, Distributed Control, Reliability, Heterogeneity, Efficiency를 들 수 있다.

4. Distributed Operating System

Network Operating System과 Distributed Operating System의 구별은 명확하다 할 수 없지만 대체로 지원하는 Distributed Operating System은 다음과 같다.

ACCENT	ArchOS
AEGIS	CHORUS
CAP	DCS
CLOUDS	Domain Structure
DEMOS/MP	Fully DP
EDEN	LOCUS
HXDP	MICROS
Medusa	RIG
MIKE	STAROS
ROSCOE	WEB
TRIX	Grapevine
Conanet	TEAMNET
NETIX	

Berkeley의 enhancement
Purdue의 implementation
Newcastle Connection
UNIX Version 8

5. LOCUS

LOCUS는 미국 UCLA대학에서 1982년에서 1983년의 2년에 걸쳐 개발된 Distributed Operating System이며, Network wide file system을 사용하면서 사용자에게는 single system에서 사용하는 것과 같은 compatibility 및 transparency를 제공하며, file의 replication을 통하여 시스템의 performance를 증가시키고 있다.

LOCUS의 가장 중요한 특징인 distributed file system은 UNIX file system과 같이 single tree structured naming hierarchy를 갖고 있지만 다음과 같은 세가지 기능이 추가되었다.

첫째, local file system내에 있는 file뿐만 아니라 remote system의 file system도 network를 통해서 transparent하게 access한다.

둘째, file을 여러 시스템에 replicate 시키며, 이 replicated된 file들의 내용을 모두 꼭같이 일치시키는 일과 가장 최근에 수정된 file의 내용을 사용자에게 제공한다.

세째, replicate된 file들에 대한 synchronization기능을 support하여 「multiple reader, single writer」의 기능을 가능하게 한다. 즉, 여러 deplicate된 file 중에서 한 file을 수정하여도 임의의 한 machine에서 그 file을 read할 때

는 수정된 file의 내용을 read할 수 있도록 한다.

이러한 file replication의 효과는 local file을 access할 때는 기존의 UNIX file system과 같지만, remote system을 access할 때는 종전의 network protocol에 의한 remote file transfer나 virtual terminal에 의한 file access에 비하면 훨씬 빠르고 편리하다.

File의 replication은 performance를 높여주지만 consistency의 문제를 발생시키므로 한 file을 여러 machine에 copy시켰을 때 한 machine의 file을 수정한다면 그 machine뿐만 아니라 다른 여러 machine에 위치하고 있는 file의 내용을 모두 똑같이 일치시켜야 하는 문제가 발생한다. file의 replication은 여러 machine에 같은 내용을 copy할 수 있게 하며, replication되는 정도는 사용하는 상황에 따라 증가할 수도 있고 감소할 수도 있도록 하며, 사용자나 application program은 replication에 대하여 transparency를 갖도록 한다. 또한 Replicate된 file들이 primary와 backup의 관계를 유지하지 않고 동등한 입장에서 처리된다.

LOCUS에서는 UNIX의 file system 대신에 file group이라는 용어를 사용하고 있으며, 이 file group의 자료 구조는 UNIX의 file system과 같이 file의 내용이 담겨 있는 데이터 블럭과 각 file을 관리하기 위한 inode, 그리고 전체 file group을 관리하는 super block으로 구성되어 있다. 그리고 각 file group은 mount 됨으로써 다른 file tree의 한 subtree가 된다. 한 logical file group은 여러 개의 physical container에 위치하며, 이 physical container를 구별하기 위해서 pack number가 사용된다.

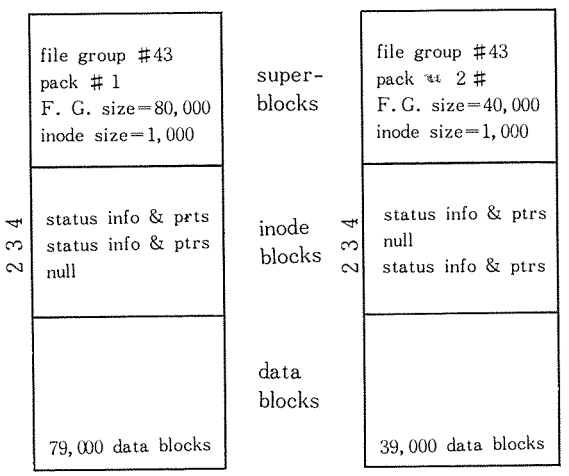


그림 3 두 Physical Container에 있는 한 Logical File Group의 모습

그림 3에서 보면 file group은 번호는 같지만 pack #는 구별되어 있으며, inode의 size는 같지만 inode block내의 내용은 다르다. 그림에서 inode 2는 두 pack에 동시에 replicate되어 있으며, inode 3는 pack # 1에 inode 4는 pack# 2에만 위치하고 있다. inode block에 inode의 내용이 담겨있는 존재 여부는 그 inode에 해당되는 file이 그 pack에 replicate되어 있는지 아닌지에 달려 있다. 이때 file이 존재하지 않는 inode는 완전히 없애버리지 않고 대신에 null로 비워 놓는다. 그 이유는 여러 machine에 replicate된 각 file이 같은 inode number를 갖게 하여 모든 file을 같은 inode number (or file descriptor)로서 access하기 위함이다. 즉, file은 그 file이 위치한 machine에 관계없이 다음과 같이 access할 수 있다.

(logical file group #, inode #)

Remote machining을 access하기 위한 방식에는 Remote Procedure Call을 사용한다.

File을 처리하는 입장에서 볼 때 각 machine은 다음과 같은 세가지 site로서 구별될 수 있다. US (Using Site)는 file을 open하여 access하기를 원하는 site이며, SS (Storage Site)는 file의 copy가 존재하는 여러 site 중에서 file의 내용을 제공하기로 선택된 site이며, CSS (Current Synchronization Site)는 file의 global access synchronization policy를 결정하는 곳으로서 US에게 SS를 선택하여 제공한다.CSS는 US가 요구하는 file이 존재하고 있는 site와 그 file에 대한 최근 version이 위치하고 있는 site를 알고 있으며 이런 정보를 바탕으로, SS를 선택한다. CSS는 communication이 가능한 site들의 집합(partition)마다 하나씩 존재한다.

6. Newcastle Connection

분산 운영 체제는 exception reporting 면도 포함해서 기능적으로 그 분산 system을 구성하는 단일 system과 동일하며, fault tolerant system은 일반적인 fault tolerant system으로 만들어져야 한다는 recursive한 기준하에 design된 분산 운영 체제로서 Newcastle University에서 개발되었으며, LNCUS와 같이 UNIX의 mount기능의 확장으로, system들을 mount 시켜 Network상에 연결된 UNIX system 들의 Naming structure를 하나로 만들므로, Network Environment에서도 전체가 하나의 system인 것처럼 쓸 수 있도록 했다.

User command와 System call이 single UNIX system에서와 동일하며, 이러한 transparency 는 user program과 non-resident UNIX software가 UNIX kernel의 system call을 할 때, 그 사이에 삽입된 Newcastle Connection Layer가 remote access를 filter out해서 Remote Procedure Call을 하고 그 결과를 user에게 줌으로서 user나 kernel위에 있는 software에서는 single system과 차이가 없게 된다.

UNIX United는 Newcastle Connection에 의해 연결된 UNIX machine의 집합적인 명칭이며 각 기계는 자신의 storage와 peripheral device들, accredit된 사용자의 set와, system administrator등이 있는 standard UNIX system의 상호 연결된 형태이다.

각 component system인 UNIX machine 의 naming structure들이 join되어 전체 system이 single naming tree를 구성하며, 각 UNIX system은 directory로 취급되며, 각 UNIX system에서 network에 연결되어 있는 어떠한 device, file, command, directory도 이용 가능하다.

UNIX United는 구성되는 각 component system의 사용자의 set, 사용자의 group, 그리고 사용자의 password file, 각 system의 system에 administrator (super-user)를 허용한다. 각 system은 그 system에 login하려고 하는 사용자를 user identifier와 password를 이용해서 authenticate할 책임이 있다.

처음 어느 system에 login한 사용자는 다른 system을 이용하고자 할 때 또 다시 그 system에 login하지 않도록, system사이에 사용자의 identifier, group identifier, password까지 이용해서 mapping하는 기능이 system 내부에 있어야 한다.

그리고 서로 organization 또는 administration이 다른 system사이에 super user에서 다른system의 super user로 mapping되지 않도록 하는 security measure에 대한 고려도 있어야 한다.

UNIX United의 naming structure는단지naming issue에 국한해서 UNIX system들 끼리의 상호관계를 나타내는 방법이며, 필요에 따라 department 또는 computer system을 쓰는 organization의 환경을 반영할 수도 있다. 그렇지만 naming structure는 바탕을 이루는 communication network의 topology를 나타내는 것은 아니

UNIX United의 service를 가능하게 하는 것은 Newcastle Connection을 이루는 Software layer에 의한 것이며, 이 newcastle connection은 resident UNIX kernel 위에 위치하며 동시에 operating system의 나머지와 사용자 program아래에 존재한다.

이 layer의 위에서는 이 layer의 기능이 kernel의 기능과 구별이 안되며, layer의 아래에서는 보통의 user process와 동일하게 보인다.

layer의 역할은 다른 System으로 redirect해야하는 System call을 가려내고, 다른 system에서 보내온 System call을 받아들이는 역할을 한다.

여러 system의 Connection사이의 통신은 Remote Procedure Call (RPC) Protocol에 의한다.

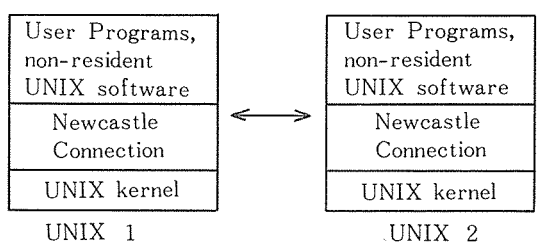


그림 4 Connection Layer

사용자의 request remote procedure call에 의해 remote site의 server를 제공하는, network상에서 well known address로 listen하는 USAM이 필요에 따라 server를 제공하고, server의 signal과 control을 수행한다. 그리고 server와 교신중에 crash가 일어나서 server와 교신이 끊기는 경우 원래의 server는 orphan이 되어, 이를 spawn한 USAM이 orphan 이라고 판단될 때 orphan이 된process를 kill시킬 수 있는 방책이 있어야 한다.

Newcastle connection의 serivce를 쓰고자 하는 program이나 command가 수행이 되면, local과 remote에 있는 object를 지칭하는 naming scheme이 있어야 한다.

이는 file open할 때, locate하는 과정이 있어서, local이면 원래의 open system call을 수행시켜주며, remote라고 판단되면, remote site의 USAM에게 request를 보내서, server를 제공받은 다음, remote site의 server에게 open system call을 위한 remote procedure call을 보내어서 되돌아온, file descriptor를 return한다.

Open된 file에 대해 read할 때는 read하기 전에 local인가 확인하고 remote이면 이미 제공받은 server에게 read를 위한 remote procedure call을 보낸다. 그렇지 않고, local일 때는 read system call을 실시한다.

7. Conclusion

분산 시스템의 개요와 그 issue, research area에 대해 알아보고, LOCUS와 Newcastle Connection 두 시스템에 대해 간략한 관찰을 해 보았다. 고속의 Local Area Network을 이용해서, tightly coupled된 시스템을 단순한 file 전송과 remote login의 service만 제공할 것이 아니라 remote file을 transparent하게 access 한다든지 programming environment를 분산시스템을 이용하도록 확장하는 것이 바람직할 것이다. 이러한 service를 제공하는 기법들은 곧 Local Area Network상에 file server를 이용하는 Workstation의 개발에도 이용될 수도 있을 것이며 더욱 human friendly한 interface를 사용자에게 제공할 수 있는 길이 될 것이다.

Reference

1) D. R. Brownbridge, L. F. Marshall, and B. Randell, "The Newcastle Connection or UNIXes of the World Unite!," Software-Practice and Experience (Vol. 12), 1982.

2) Bob Lyon, Gary Sager, and members of the SUN NFS project, "Overview of the Sun Network File System," January 1985.

3) Bruce Walker, G. Popek, Robert English, Charles Kline, and Greg Thiel, "The LOCUS Distributed Operating System," ACM, University of California at Los Angeles.

4) John A. Stankovic, "A Perspective on Distributed Computer Systems," IEEE Transactions on computer (Vol. C-33, 12) December 1984.