VDFS에서 원격 화일 서비스의 성능 향상 알고리즘

윤 동 식† 이 병 관††

요 약

본 논문에서 구축된 VDFS는 소형 시스템내에서 현존하는 NFS와 RFS 같은 대칭형 화일 시스템 구조들에서 나타나는 문제점들을 해결한 비대칭형 분산 화일 시스템을 제안하였다. 특히 본 논문에서는 중점적으로 다루어질 부분은 공유 화일의 중복 사용을 통해서 보다 많은 투명성(Transparency)을 제공하고 캐쉬 기반으로 한 원격 화일 서비스를 한층 더 증가시켜 서버 부하와 통신량을 감소시키므로 시스템의 성능을 향상 시키는데 있다.

Performance Enhancement Algorithm for Remote file service in VDFS

Dong Sic Yun† and Byoung Kwan Lee††

ABSTRACT

VDFS(V-kernel Distributed File System) constructed in this paper proposes an asymmetric Distributed File System so that various problems of existing symmetric structures like NFS & RFS may be solved in small-sized system. Particulary, in this paper, server workload can be reduced by using a caching mechanism and remote access transparency can be improved by using a sharing file duplicately.

1. 서 론

분산 처리 시스템에서 사용되는 운영체제를 크게 네트워크 운영체제(Network operating system: NOS), 분산 운영 체제(Distributed Operating System:DOS) 나눌 수 있다.

NOS는 기존의 중앙 집중식 처리 시스템 운영 체제에 네트워크 운영 체제를 이용해서 사용자간에 서로 통신을 할 수 있는 시스템으로 각각의 사용자 컴퓨터들에는 서로 다른 운영 체제를 가지고 있게 된다. 분산 운영 체제는 기존의 중앙 집중식 처리 방식이 아닌 하나의 운영 체제를 각각의 사용자 컴퓨터들에 분산을 시킨 후에 처리하는 방식이다. 그러므로 분산 운영 체제는 공유되어야 할 화일들은 각각의 사용자 컴퓨터에 공히 분산을 시키고 독립적인 성격을 띤 화일들은 각각의 성격에 맞는 컴퓨터에만 설치 하여서 사용하는 형태이다.

분산 운영 체제에서 얻을 수 있는 장점으로는 사용자의 위치에 관계없이 시스템을 이용할 수 있는 투명성(transparency), 안정성, 서버 시스템 고장의 복구, 처리 용량, 유연성, 단순성, 신뢰성 등이 보장된다는 것이다. 지금까지의 분산 운영 체제의 예로는 ACCENT[10], LOCUS, CHORUS [11], MOS[12], V-SYSTEM[13], IBIS[14], DUNIX[15], Andrew, Cedar[16], PULSE[17], CODA, NFS, RFS 등이 있다. 분산 화일 시스템의 주 목적은 물리적인 컴퓨터의 기억장소와 데이타의 공유를 사용자들에게 허락하는 것이다.

본 논문은 분산 운영 체제(Distributed operating system)환경하에서 기존의 비대칭형 분산 화일 시스템인 Sprite, Andrew, Cedar, IBIS, PULSE, CODA, LOCUS, ACCENT, V-SYSTEM의 각각의 화일 구조에서 나타나는 장·단점을 찾고 이를 수정 보안하여 시스템의 성능, 안정성, 그리고 투명성을 증가시키고 원격 화일 서비스(remote file Service)의 처리 속도를 증가시킨 화일 구조를

† 정 회 원 : 아세아 항공전문학교 정보통신공학과 교수
†† 정 회 원 : 관동대학교 전자계산학과 부교수
논문접수 : 1994년 12월 2일, 심사완료 : 1995년 3월 21일

설계하고 구현 방안을 제시하였다. 특히, 이 논문에서 중점적으로 다루어질 부분인 캐싱을 기반으로 한 원격 화일 서비스을 한층 더 증가시켜 서버 부하와 통신량을 감소시킴으로써 시스템의 성능을 향상 시키고, 비대칭형 구조가 대형 시스템 위주로 설계되어있는 것을 소형 시스템에 맞게 설계하였다. 본 논문에서 설계된 File System을 V-Kernel 하에서 설계하였으므로 VDFS(V-Kernel Distributed File System)이라 칭한다.

2. 분산 화일 시스템의 기본 구성 요소

분산 시스템은 처리 장치, 기억 장치, 입·출력 장치 등의 기본적인 처리 능력을 가진 독립적인 장치(Autonomous computing machine)들을 통신으로 연결하여 구성한 종합적인 컴퓨터 시스템을 말한다. 분산 시스템은 다중성(Multiplicity), 메세지 전송(message passing), 지역 자동화(local autonomy), 투명성과 같은 요구 조건들을 만족하는 컴퓨터들의 집합이라고 할 수 있다.

이와 같은 특성을 만족하는 분산 시스템은 시스템의 향상, 시스템의 안정성 및 신뢰성의 향상, 자원 공유의 용이함 등의 장점을 가진다.

분산 처리 시스템의 한가지인 네트워크 시스템은 각 노드가 자신의 운영 체제를 가지고 있어 시스템의 투명성을 거의 제공하지 못한다. 그러나 분산 운영 체제 시스템은 전체를 관장하는 하나의 운영 체제만이 존재하여, 사용자는 마치 하나의 컴퓨터를 사용하는 것과 같은 느낌을 받는다.

시스템의 분산과 투명성을 고려한 분산 운영 체제와 네트워크로 연결된 네트워크 운영체제를 비교하면 아래의 〈표 1〉과 같다.

위에서 살펴본 바와 같이 분산 운영 체제는 자원들이 모두 분산 및 중복되어 있고 또한 이러한 지리적으로 분산된 자원들을 투명하게 사용할 수 있으므로 완전한 자원의 공유와 투명성이 가능하다고 할 수 있다[18].

분산 화일 시스템은 한 머신상에서 클라이언트(Client)의 요구를 수행하는 프로세서인 서버(Server), 정의된 인터페이스(interface)를 사용하여 서버에게 서비스를 요구하는 클라이언트, 그리고 서버에서 클라이언트가 요구한 작업을 요구에 맞게 처리하는 행위인 서비스(service)의 세가지로 구성되어진다.

분산 화일 시스템은 위의 기본적인 구조를 바탕으로 투명성, 전역 접근 열람(glo bal access view), 이름명시(naming), 공유 시멘틱스(sharing semantics), 원격 화일 서비스(remote file service), 확장성 등의 여러가지 기능들을 제공한다.

〈표 1〉 NOS와 DOS
〈Table 1〉 NOS & DOS

	N O S	D O S
자원공유	가 능 함	가 능 함
신 뢰 도	낮 음	높 음
가 용 성	없 음	높 음
처 리 율	낮 음	높 음
통 신 량	많 음	적 음
투 명 성	불 가 능	가 능

첫째, 투명성은 사용자들에게 하드웨어 환경과 소프트웨어의 복잡한 작업 및 특수한 상황을 인식할 필요가 없도록 지원한다. 분산 화일 시스템에서 일반 사용자들은 여러 시스템으로 구성된 화일 시스템을 단일 시스템에서 사용하는 것처럼 느끼게 지원한다. 분산 화일 시스템에서 접근(access), 위치(location), 중복(replication), 그리고 결함 허용(fault-tolerance)들에 투명성이 제공되어야 한다.

둘째, 이름명시(naming) 작업은 디스크에 저장되어 있는 화일의 위치를 알아내기 위한 방법으로써 기존의 단일 화일 시스템에서는 디스크 주소(disk address)만으로 명시되었지만 분산 화일 시스템에서는 화일 디스크 주소에 특정한 머신 주소(machine address)까지 첨가되어 명시되어야 한다.

네트워크상에서 사용자가 화일이 실제 저장된 위치를 알 필요가 없도록 위치 투명성(Location transparency)을 제공하여야 한다.

분산 화일 시스템의 설계시 고려되어야 할 화일의 위치 개념과 이름명시 형태에는 화일 위치 개념, 이름명시 형태가 있다.

세째, 화일 구성에 대한 전역 접근 열람(global access view)은 여러 노드의 사용자들이 분산된

화일들을 마치 하나의 계층적 구조처럼 느끼게 하는 것이다. 전역 접근 열람은 이름명시 기법을 결정하는데 중요한 기준이 되며, 또한 사용자의 이동(mi gration)을 용이하게 한다.

네째, 공유 시맨틱스는 여러 사용자들이 공유되는 화일이 open되고 close되는 한 세션(session) 동안에 클라이언트들과 서버 사이에 정의된 인터페이스로서 화일 시스템의 성능 및 일관성을 평가 하는데 있어 중요한 기준이 된다. 공유 시멘틱스(shared semantics)는 UNIX 시멘틱스(semantics), 세션 시멘틱스(session semantics), 그리고 불변 공유 화일 시멘틱스(immutable shared files semantics)로 분류할 수 있다.

다섯째, 원격 화일을 서비스하기 위한 방법으로는 신뢰성을 중시하는 원격 호출 서비스 방법(remote call service method)과 시스템 성능을 중요시하는 캐싱(caching)을 사용한 두가지의 방법이 있다.

아래의 〈표 2〉는 원격 서비스 방법과 캐싱 방법의 장·단점을 나타낸 그림이다[18].

〈표 2〉 원격 서비스와 캐싱
〈Table 2〉Remote Service & Caching

	원 격 서 비 스	캐 싱
목 적	신뢰성을 중시한 원격 서비스	통신량과 서버 부하의 감소
장 점	서버 노드의 불의의 데이타 손실을 막을 수 있다.	통신량과 서버 부하 감소로 처리 속도 증가
단 점	응답 결과가 메세지화되어 송수신 횟수로 통신량 증가 및 상과 병목현상 발생	설계적인 면에서 전송이 지연되며, 일관성 문제가 커짐

캐싱 방법은 원격 화일에 대한 요구가 있을 시에 우선 캐쉬안에 해당 데이타가 있는지를 검색하고 존재하지 않을 때에만 클라이언트는 필요한 데이타를 서버에게 요청한다. 데이타를 수신 받은 클라이언트는 데이타를 캐쉬안에 넣은 후에 화일에 대한 서비스를 한다.

캐싱과 원격 서비스 사이의 선택은 성능의 증가와 단순성의 감소 양 측면에 있어서 상당한 상호 배반성(trade off)을 갖는다. 우리는 이러한 상호 배반성를 두 가지 면에서 장점과 단점을 열거하면서 평가하고자 한다.

많은 원격 접근들은 캐싱이 사용될 때 지역 캐쉬에 의해서 효율적으로 처리될 수 있다. 화일 접근 형태에서 지역적 이용은 캐싱을 더욱 가치있게 한다. 그러므로 대부분의 원격 접근들은 지역 접근과 마찬가지로 빠르게 제공될 수 있다. 더구나 서버와의 접촉도 각각의 접근보다도 오히려 드물다. 결과적으로 서버 부하와 네트워크의 통신량은 감소되며, 확장성에 대한 잠재성은 증가된다. 대조적으로 모든 원격 접근을 원격 서비스 방법이 사용될 때 네트워크를 통해서만 처리된다. 네트워크 통신량과 서버 부하, 그리고 성능에 있어서 발생되는 문제점은 분명히 존재한다.

캐싱에서 실행되는 큰 자료를 전송하는데 따른 전체 네트워크상의 부담은 원격-서비스의 방법에서와 같은 특정 요구에 대한 연속적인 응답들이 전송되는 부담보다는 적다.

만일 요구들이 임의 디스크 블럭이라기 보다는, 항상 크고 연속적인 자료의 시멘틱이라는 것을 알고 있다면, 서버상에서의 디스크-접근 루틴은 훨씬 최적화될 수 있다.

캐쉬-일관성 문제는 캐싱의 가장 큰 결점이다. 쓰기 접근 빈도가 적은 접근 형태에서 캐싱은 아주 우수하다. 그러나 쓰기의 빈도가 많을 경우, 일관성 문제를 해결하기 위해 사용된 기법들은 성능과 네트워크 교통량, 그리고 서버 부하에 있어서 심각한 부담을 초래한다.

캐싱의 장점을 살리기 위해서는 실행을 지역 디스크 또는 주기억 장치를 가지고 있는 기계상으로 이동시켜 행해야 한다. 디스크가 없는 소량의 메모리를 가지고 있는 기계들은 원격 서비스 방법을 통해서 실행되어야 한다.

캐싱에서 자료들은 화일 명령의 특정 요구에 대한 응답이기 보다는, 오히려 서버와 클라이언트 사이의 메시지로 전송되기 때문에 낮은 수준의 기계간 인터페이스는 고수준의 사용자 인터페이스와는 아주 다르다. 반면에 원격-서비스 모형은 네트워크를 통해 연결된 지역 화일 시스템의 확장이라 할 수 있다. 따라서 기계 상호간의 인터페이스는 지역 사용자 화일 시스템 인터페이스가 투영되었다고 할 수 있다.

여섯째, 확장성은 Andrew와 같은 대규모 시스템 설계시 고려해야 할 사항으로 서버에게 서비스의 양을 제한 하는 것으로 서버마다 처리할 수 있는 자원을 고정적으로 할당하여 클라이언트가 자원을 요구할 경우 특정한 서버에게 요청한다.

브로드 캐스팅 방법(broad casting method)은 네트워크 상의 모든 머신이 참여해야 하기 때문에 통신상 오버헤드가 크므로 대규모의 시스템에는 적합하지 않은 방법이다.

노드가 비정상일때나 네트워크의 두절시 시스템을 복구하기 위한 방법으로 무국적 프로토콜(Stateless Protocol)을 사용하는 것이다. 무국적 프로토콜은 서버가 클라이언트의 비정상적인 상태을 감지할 필요가 없을 뿐더러 클라이언트의 정보도 가지고 있을 필요가 없으므로 확장시에 유리하다.

확장 가능한 시스템을 구축하기 위해서는 중앙 집중 통제 형태와 중앙 집중적 자원을 사용해서는 않되고, 자원의 클러스트링(clustring)이 필요하다.

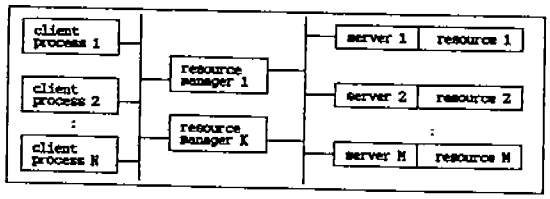
3. VDFS의 시스템의 설계

현재까지 연구된 분산 화일 시스템에는 Andrew, Sprite, CODA, LOCUS, NFS, RFS등이 있다. 본 논문에서 설계된 분산 화일 시스템은 종전의 시스템에 비해 전역 접근 열람, 투명성 및 원격 화일 서비스의 처리 속도 등과 같은 기능 제공에 있어 성능이 향상되었다.

NFS 혹은 RFS의 단점인 전역 접근 열람의 부재, 투명성의 미흡, 원격 화일 서비스의 처리 속도 저하를 기존의 Andrew, Sprite, CODA, LOCUS 등의 구조에서 사용한 기법을 응용하여 보완한 시스템이다.

이 장에서는 VDFS의 설계 목표와 구조에 관해서 기술하고, VDFS의 설계 및 구현 방안을 제시하고자 한다.

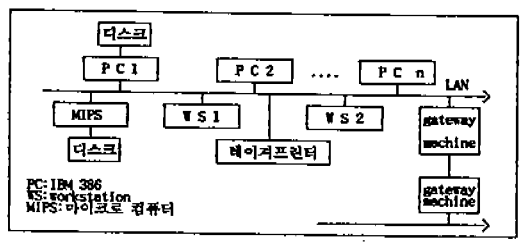
VDFS 시스템에서는 비대칭형 시스템인 Andrew, Sprite, CODA, LOCUS 등의 장점을 이용하여 화일 구성에 대한 전역 접근 열람을 제공하여 원격화일을 단일 화일 시스템에서 지역 화일과 같이 사용할 수 있게 하고, 기존의 단일 시스템에서 사용하였던 프로그램을 변경없이 그대로 이용하는 접근 투명성과, 사용자가 화일의 위치를 독립적으로 화일 접근이 가능하게 한 위치 투명성, 읽기만 가능하게 하여 사용자에게 화일에 대해 최대한의 유용성을 제공시키는 중복 투명성 제공과 임의의 사용자가 원격 화일을 사용하고자 할 경우 원격 화일은 캐싱하여 사용되어지므로 통신상의 서버 부하를 줄일 수 있으므로 원격 화일 서비스의 성능 향상을 가져오며, 지역 캐쉬 메모리 장치를 사용함으로써 다른 시스템 성능에 영향을 최소로 줄일 수 있는 반면 사용자들은 최대한으로 자율성을 보장할 수 있다. (그림 1)은 분산 자원 관리도이다.



(그림 1) 분산 자원 관리
(Fig. 1) distributed resource management

3.1 시스템 설계 환경

VDFS의 운영 환경은 소형컴퓨터에서 이기종간에 이루어지는 분산 화일 시스템을 구축하기 위하여 다수의 퍼스널 컴퓨터(PC)와 워크스테이션(workstation) 그리고 RISC방식의 CPU를 사용한 마이크로 컴퓨터인 MIPS을 IEEE 802 Ethernet을 사용한 LAN에 의해 상호 연결 하였다. 이들 다수의 기계들은 V SYSTEM UNIX를 운영체제로 사용했다.



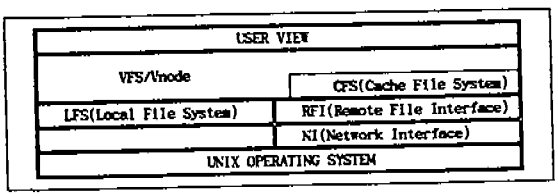
(그림 2) VDFS의 운영 환경
(Fig. 2) VDFS Operating Environment

3.2 VDFS 시스템 구조

지역적으로 분산되어 있는 컴퓨터들을 이용하여 분산처리 시스템을 구성하므로 분산 화일 시스템에서 발생되는 혹은 시스템 설계시 고려되어

야 할 사항은 다음과 같다. 첫째, 각각의 분산된 컴퓨터에 할당된 화일 시스템 구조의 차이와 기존의 화일 시스템과의 compatibility유지에 고려를 해야 한다. 둘째, 시스템을 사용하는 사용자들에게 분산된 화일을 마치 자신의 머신에서 사용하는것 같이 원격 처리 호출(remote procedure call)시에 반드시 투명성(tranparency)이 제공되어야 한다. 셋째, 각각의 머신을 연결시키는 통신 장비와 논리적으로 연결시키는 프로그램이 제공되어야 한다. 넷째, 화일 시스템의 설계에서 중시되어야 할 점은 화일 시스템 사용시 신뢰도 및 효율성이며, 화일 시스템의 확장성 또한 고려되어야 할 사항들이다. 위에서 살펴 본 고려사항들을 바탕으로 하여 화일 시스템 구조를 설계하였다.

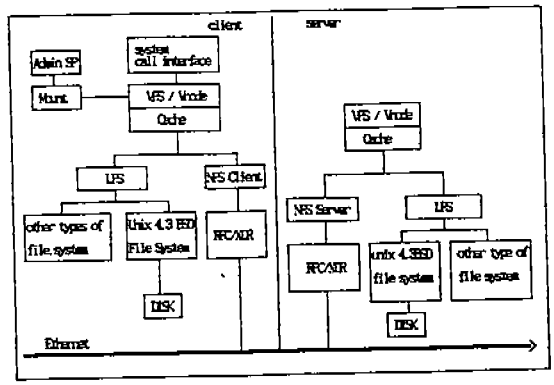
본 화일 시스템의 논리적인 구성요소에는 원격 화일 시스템에 있는 화일을 지역시스템에 있는 화일과 같이 호출할 수 있도록 하는 RPC(Remote Procedure call)와 데이타 전송에 있어서 내부 데이타를 통신될 수 있는 형태로 변환하여 전송할 수 있는 형태로 만들어 주는 XDR(eXternel Data Representation), 그리고 노드들의 상태와 네트워크의 상태를 일정한 시간마다 모니터링하여 RFI에게 알려 주는 daemon 프로세스인 상태 모니터(status monitor)와 화일들의 lock 정보와 캐싱에 사용되는 페이지를 관리하는 Lock 관리자(Lock manager), 페이지 관리자(page manager)등으로 구성되었다.



(그림 3) VDFS 논리적인 계층구조
(Fig. 3) VDFS Logical Hierarchial Structure

다음의 (그림 4)에서 보는 바와 같이 클라이언트와 서버 부분으로 시스템을 구분 지을 수 있다.

사용자로부터 화일의 요구가 있을 시에는 클라이언트의 VFS/Vnode는 화일 요구에 따라 지역 화일과 원격 화일로 구분지어진다. 이 구분에서 지역화일인 경우에는 LFS에 의해 수행되어지고 그러지 않은 경우에는 NFS에 의해 수행되어 지게 된다. NFS는 일단 먼저 사용자가 요구하는 화일을 캐쉬 메모리에서 검색이 이루어진 후에 캐쉬 메모리 상에서 요구된 화일의 페이지를 가지고 RFI로 넘어간다. RPC/XDR은 수신한 함수의 인자를 메세지화 하여 네트웨을 이용하여 서버로 보내진다. 메세지를 받은 서버의 RPC/XDR는 서버의 RFI의 해당 시스템을 호출(invoke)하여 인자를 할당한다. 호출된 시스템 콜은 서버의 VFS/Vnode를 통해 지역 화일 시스템인 NFS를 사용하여 화일 요구에 대한 처리를 한 후 결과를 역으로 사용자에게 반환한다.

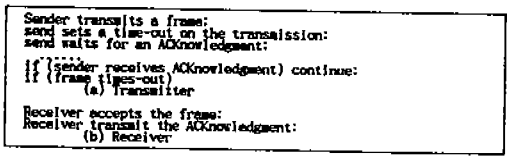


(그림 4) VDFS 구성도
(Fig. 4) VDFS Configuration

3.3 프로세스간 통신 프로토콜의 설계

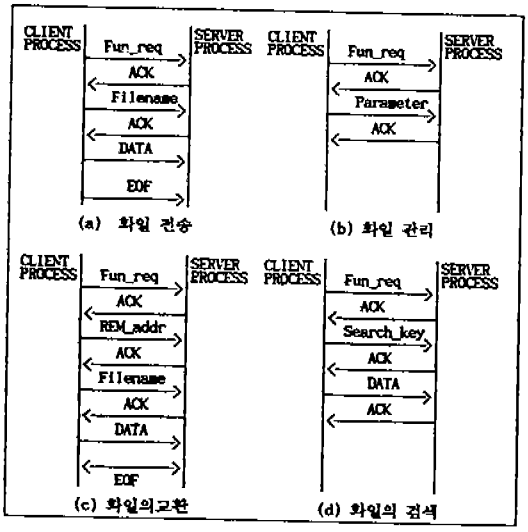
프로세스간의 통신은 기본적으로 두 프로세스가 정해진 절차에 따라 일정한 형식의 메세지를 주고 받음으로써 실현되는데 본 시스템에서는 클라이언트 프로세스와 서버 프로세스로 구분하여 설계하였다. 클라이언트 프로세스는 사용자의 요구에 따라 원격의 서버 프로세스에게 해당 기능을 요청하며 서버 프로세스는 원격의 클라이언트 프로세스가 보낸 요청에 따라 필요한 기능을 수행하는 역할을 수행하도록 하였다. 시스템의 각 기능을 수행하기 위하여 클라이언트 프로세스와 서버 프로세스간에 이루어지는 프로토콜은 다음과 같다. 또한 두 프로세스간의 교착 상태를 방지하기 위하여 클라이언트와 서버가 기능 수행에 필요한 변수 등을 주고 받는 과정에서는 stop-

and-wait방식의 프로토콜을 사용하여 실제 데이타를 전송하는 단계에서는 ACK를 기다리지 않는 프로토콜을 사용하여 시스템 성능을 향상 시켰다.



(그림 5) stop-and-wait 흐름 제어
(Fig. 5) stop-and-wait Flow control

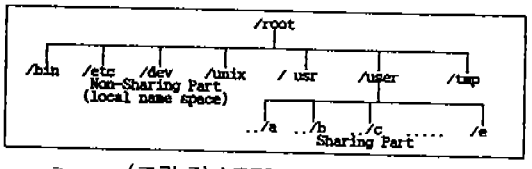
(그림 6)은 클라이언트와 서버간의 프로토콜을 나타낸 그림이다.



(그림 6) 클라이언트-서버 프로토콜
(Fig. 6) Client-Server PROTOCOL

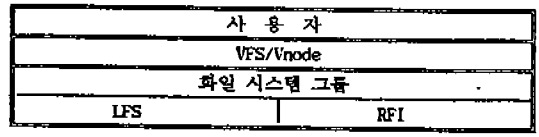
3.4 정 책

원격 마운트 구조는 UNIX와 동일한 접근 열람을 제공하고 원격 화일의 공유를 용이하게 할 수 있다.



(그림 7) VDFS 화일 구조
(Fig. 7) VDFS File Structure

전역 접근 열람을 사용자에게 제공하기 위하여 VDFS에서는 계층형 화일 구조로 공유하는 부분과 공유되지 않는 부분으로 나누었다. 그 이유로는 첫째, 모든 노드에서 모든 부분을 UNIX화일 시스템과 동일하게 공유하기 어렵기 때문이다. 둘째, 계층적 화일 구조가 처음 구성된 후 /bin, /etc, /dev와 같은 디렉토리는 거의 변경이 없이 사용자에게 사용되며, /tmp 디렉토리는 시스템 프로그램에 의해 빈번히 변경되나 사용자의 공유 관심 대상이 아니므로 공유할 필요가 없다. 세째, /user와 같이 사용자 영역을 나타내는 디렉토리는 사용자에 의해 빈번히 read · write가 수행되므로 모든 노드에서 공유하여 내용의 변경을 즉시 반영하여야 한다.



(그림 8) VFS/Vnode interface 위치
(Fig. 8) VFS/Vnode interface Position

분산 화일 시스템에서 화일의 유용성을 높이기 위해서는 화일의 중복과 노드의 자율성을 최대한 보장하는 것이 필요하다. VDFS에서는 read-only 화일에 대해서는 중복 허용을 허락하고 자주 사용되는 화일은 지역 캐쉬 메모리에 중복하여 저장한 후 차후에 요구가 있을 시에 캐쉬에 저장된 화일을 인출하여 사용한다.

3.5 VFS/Vnode 인터페이스

VFS/Vnode 인터페이스는 구성된 화일 시스템과 커널 사이의 잘 정의된 인터페이스 제공 원격 화일 사용시 서버들의 커널을 사용할 수 있도록 하는 인터페이스의 정의, UNIX화일 시스템과 거의 동일한 액세스 시멘틱스 지원, 모든 화일 시스템 오퍼레이션의 자동화, 기존의 단일 화일 시스템을 기본으로 한 비교적 간단한 원격 화일 구현, 단일 시스템에 비해 최소한의 분산 화일 시스템 성능 저하 같은 목표를 기본으로 다수의 화일 시스템을 동시에 지원하기 위하여 설계 개발되었다.

기존의 VFS/Vnode 인터페이스는 위에서 언급한 목적을 달성하는 과정에서 다수의 도메인을 거쳐 화일을 lookup 할 경우 성능상에서 VFS/Vnode층의 삽입으로 인하여 기존의 단일 시스템에 비해 lookup 시간이 많이 걸린다. 또한 다수의 도메인을 거쳐 화일을 lookup할 경우 VFS층과 Vnode층이 함께 필요하므로 복잡하고 lookup 시간이 많이 걸리는 문제점이 발생하게 된다. 이를 해결하기 위하여 Vnode의 스택화 방법을 이용함으로 해결하였다.

Vnode스택은 여러개의 화일 시스템을 거쳐 lookup할 경우에 발생되는 오버헤드를 제거하기 위해 제안된 것으로 lookup시 VFS층의 필요성을 제거하고 스택을 사용한 Vnode를 하나의 Vnode로 취급한다. Vnode의 스택화는 여러 도메인을 거쳐 화일을 lookup할 때 기존의 VFS/Vnode 인터페이스에 비해 VFS층의 필요성을 제거하여 VFS객체를 lock하는 오버헤드와 VFS를 검색하여 지역 루트 Vnode를 찾는 오버헤드를 줄일수 있을 뿐만 아니라 Vnode층만으로 lookup 할 수 있어 구현이 간단해진다.

Vnode 스택화를 위해서 다음과 같은 두개의 오퍼레이션이 Vnode오퍼레이션 세트(set)에 삽입되어 사용된다.

Vnode 구조를 변경한 뒤 push와 pop 두개의 오퍼레이션을 새롭게 Vnode 오퍼레이션에 첨가시킨다.

VFS/Vnode인터페이스에 의해 기존에 사용되었던 lookup, mount, 그리고 unmou nt와 같은 알고리즘들이 변경되어야 한다.

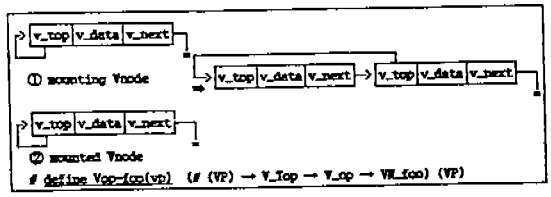
```
- PUSH(below, above)
  Below Vnode와 above Vnode를 스택 형태로 만드는 오퍼레이션이다.
  int vn_push (below,above)
  struct vnode *below;
  struct vnode *above;
  {
      struct vnode *tvp = below;
      above->v_next !=NULL)
      {
        tvp->v_top = above;
        tvp = tvp->next;
      }
  }
- POP(top)
  Top Vnode를 스택의 top에서 제거하는 오퍼레이션이다.
  int vn_pop (vp)
  struct  vnode  *vp;
  {
    struct  vnode  *bvp = vp->next;
    struct  vnode  *tvp = bvp->next;
    bvp->v_top = bvp;
    while ( tvp->next!=NULL )
    {
      tvp->v_top = bvp;
      tvp = tvp->next;
    }
    vp->v_next = NULL;
  }
```

(그림 9) Push & Pop 오퍼레이션 알고리즘
(Fig. 9) Push & Pop Operation Algorithm

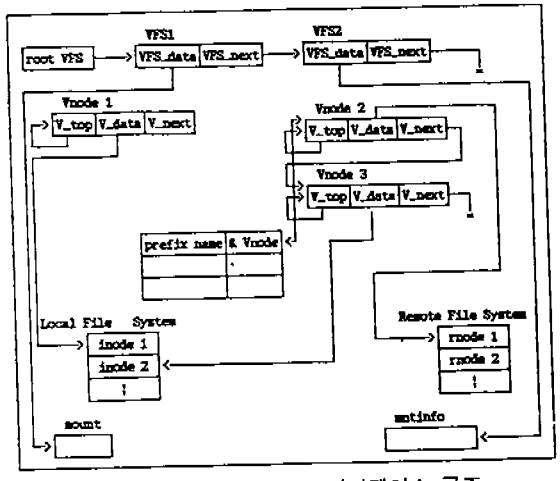
* Vnode 스택화에 의해 mloop 부분이 삭제된다.
* mount는 prefix hint를 구성하는 부분과 push 오퍼레이션이 삽입된다.
* unmount는 prefix hint에서 엔트리를 삭제하는 부분과 pop 오퍼레이션이 삽입된다.

3.6 캐싱을 기반으로 한 원격 화일 서비스

디스크 캐쉬를 사용함으로써 통신량을 감소시켜 네트워크의 서버 부하를 감소 시킬 수 있다. Andrew시스템은 화일 전체를 캐쉬한다. 그러나 본 시스템에서는 화일 전체를 캐쉬하지 않고 지역 디스크에 맞게 개별적인 블럭 캐쉬를 제공한다. 캐싱의 단위가 증가되면 될수록 적중률(hit-ratio)이 증가되는 반면에 실제적인 자료 전송은 지연된다. 그러므로 본 시스템은 가장 알맞는 단위인 블럭 단위로 캐싱을 시켜 일관성 문제를 어



(그림 10) Vnode 스택
(Fig. 10) Vnode Stack



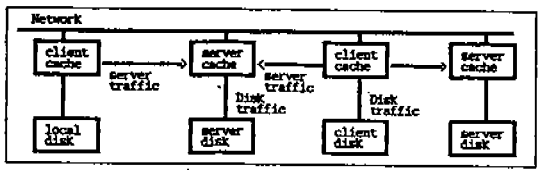
(그림 11) VFS/Vnode 인터페이스 구조
(Fig. 11) VFS/Vnode Interface Structure

느 정도 해결하였다. 캐싱되는 단위를 결정하는데 RPC 프로토콜이 사용되는 경우, 네트워크의 전송 단위와 원격 프로시듀어 호출의 프로토콜 서비스 단위가 고려된다. 네트워크의 전송 단위는 약 1.5K바이트이며, 너무 큰 단위로 캐싱되는 자료는 전송을 위해서 분해(disassemble)되며, 수신상에서 다시 재결합(reassemble)된다. 블럭 크기와 전체 캐쉬의 크기는 블럭 캐싱 기법에 상당히 중요한 요소이다. UNIX시스템에서 일반적인 블럭의 크기는 4K바이트 또는 8K바이트이다. 대량의 캐쉬의 경우 불합리하다.

다음의 그림은 각각의 클라이언트와 서버를 클라이언트 캐쉬와 서버 캐쉬를 이용한 화일들의 관리 체계이다. 이 구조는 Andrew 시스템과 같이 화일 제어기(file traffic)가 네트워크를 이용하여 각각의 서버 캐쉬와 클라이언트 캐쉬를 접근 제어하는 시스템구조이다.

3.6.1 캐싱을 위한 논리적 구조

기존의 NFS는 원격 서비스를 기반으로 실행되므로 read 및 write 시스템 콜이 RPC/XDR를 이용하여 직접 서버의 오퍼레이션을 호출하였으나 RFI의 read 및 write는 클라이언트의 원격 캐쉬 영역에서 해당 화일을 사용하므로 직접 서버의 오퍼레이션을 호출하는 것이 아니라 NI의 페이지 관리자에게 read와 write를 요청한다.



(그림 12) 화일 시스템 캐싱
(Fig. 12) File System caching

페이지 관리자는 캐쉬안에 해당 페이지가 있는지를 검색하고 있을 시에는 해당 페이지를 넘겨주고 없을 시에는 서버 페이지 관리자에게 해당 페이지를 요구한다.

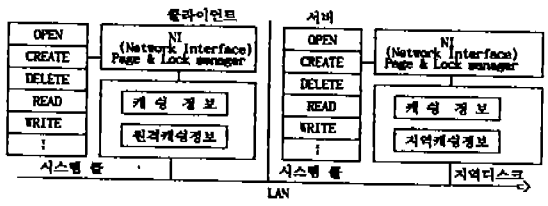
서버 페이지 관리자는 해당 페이지를 지역 캐쉬 영역에서 추출하여 클라이언트의 원격 캐쉬영역에 캐싱을 시켜준다.

3.6.2 자료 구조

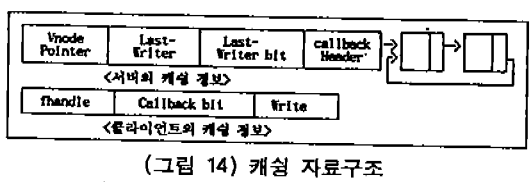
자료 구조는 클라이언트와 서버에 따라 조금 다르다 먼저 클라이언트 부분은 (그림 14)에서 보는 것과 같이 캐싱에 필요한 정보를 저장하는 자료 구조 형태를 갖는다. 첫번째 fhandle부분은 서버에서 캐쉬한 화일의 구별자를 나타내고, 두번째 callbackbit 부분은 클라이언트가 서버에 갱신한 화일을 다시 캐쉬할 필요 없이 계속 사용할 것인지 아니면 다시 캐쉬해서 사용할 것인지의 여부를 on/off형태로 저장한다. 마지막으로 write field 부분은 변경된 페이지가 서버에 반영되었는지의 여부를 on/off 형태로 나타낸다. 그리고 서버부분은 첫번째 Vnode pointer(vp) 필드 서버 화일 시스템의 해당 화일을 나타내는 화일 구별자이고, 둘째번 부분은 Last-w riter 필드로써 vp에 대한 최종 writer가 누구인지를 클라이언트의 id를 이용하여 나타내게 된다. 세번째로는 Last-writer bit 부분으로 최종 writer가 수정한 데이타를 모두 갱신 했는지 여부를 on/off형태로 표시한다.

클라이언트가 임의의 화일을 참조할 경우 서버는 클라이언트에게 callback promise를 넘겨주고 클라이언트 id를 callback header에 순환 연결 리스트 형태로 연결시킨다.

NFS 기반으로 구성된 RFI에서 캐싱 메카니즘을 기반으로 한 원격 화일 서비스 기법을 사용하기 위해서는 원격 서비스 방법에 사용하는 화일



(그림 13) 논리적 캐싱구조
(Fig. 13) Logical Cache Structture

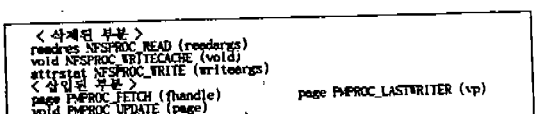


(그림 14) 캐싱 자료구조
(Fig. 14) Caching Data Strtucture

을 open 할 때 지역 노드에서만 open이 수행되고 원격 노드에서는 read · write 시스템 호출 발생시 open 시스템 호출을 수행하고, read · write 수행이 끝난 후 close한다. 그러므로 read · write 시스템 호출은 직접 RPC/XDR를 이용하는 것이 아니고 캐쉬를 사용하므로 원격 서비스를 받을 수 있다. 캐쉬에 저장된 화일인 경우에는 인출을 수행하고 그렇지 않을 경우에는 RPC/XDR를 수행하여 원격 서비스를 받는 구조이다. 서론 부분에서도 언급하였지만 캐쉬를 이용하여 원격 서비스를 받으므로 통신상의 서버 부하를 감소시킬 수 있다. 그러나 캐쉬를 사용시에는 화일의 일관성 문제가 대두되게 된다. 이 문제를 해결하기 위하여 좀더 연구해야 할 것이다.

캐싱을 사용한 원격 화일 서비스를 위하여 변형된 RPC 구조는 다음과 같다. 기존의 NFS의 RPC 구조에서 사용되어지던 read · write부분을 삭제시키고, 캐쉬에 저장된 내용을 인출하고 캐쉬에 없을 시에는 원격 호출 부분이 삽입되었다.

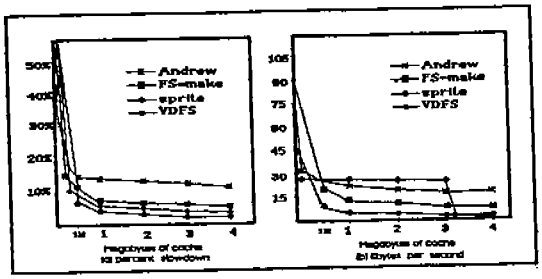
다음은 RPC의 서비스 캐쉬에 필요한 캐싱 정보를 저장하는 자료 구조이다. Dr_flag는 오퍼레이션의 수행 정보를 나타내는 변수로 IN_PROGRESS 상태와 NO_PROGRESS상태가 있다. IN_PROGRESS 상태는 동일한 오퍼레이션이 이미 수행 중임을 나타내는 것이고 NO_PROGRESS 상태는 동일한 오퍼레이션이 성공적으로 수행되었거나 도중에 에러가 난 상태를 나타낸다. Dr_xid는 RPC의 유일한 서비스 id이고 dr_addr는 네트워크 버퍼 주소이다. dr_proc, dr_vers와 dr_prog는 특정한 서버의 RPC 오퍼레이션을 호출하기 위한 구별자이다. dr_field는 오퍼레이션의 수행 결과를 나타내는 것으로 성공적으로 수행이 되었을 경우 상태(STATUS)는 OK이고 그렇지 않을 경우는 ERROR가 할당된다. dr_next는 서비스 캐쉬의 해싱을 위한 변수이다.



(그림 15) 변형된 RPC구조
(Fig. 15) Upgraded RPC Structure

위에서 설계한 클라이언트 캐쉬와 서버 캐쉬를 이용한 메모리 관리에는 Nelson et al의 "caching in the sprite network file system"의 write-sharing을 이용하여 평가하였다. 이 시스템에서 가장 중요시 되어져야 할 것은 캐쉬의 일관성 문제와 동기화 문제이다. 본 시스템의 해결 방법으로 FS_LOCK 호출을 사용하여 해결하였다.

(그림 16)은 diskless workstation에 수 Megabyte을 사용한 시스템의 벤치마크 시험결과 클라이언트 캐쉬를 사용하여 평균적으로 디스크 없는 워크스테이션 상에서 스피드 향상을 가지고 왔으며 자체 디스크를 지니고 있는 워크스테이션보다 다소 느릴 뿐이다. 또한 클라이언트 활동에 대한 서버의 활용율의 감소 효과를 가지고 왔다. 그러므로 단일 서버가 약 30~50개의 클라이언트에게 서비스를 제공할 수 있을 것이다. NFS를 기초로 한 본 화일 시스템에서 NFS의 화일 처리 속도보다 빠른 결과를 가지고 왔다. 그러나 소형 시스템에서 캐쉬 메모리를 늘릴 수 있는 한계성이 있어 이 한계성을 해결하는 것이 차후 해결되어야 할 과제로 남았다.



(그림 16) 성능 향상률 비교
(Fig. 16) Comparison of performance improvement

```
struct dupreq
  {
      enum progress     dr_flag;
      struct netbuf     dr_addr;
      u_long            dr_xid;
      u_long            dr_proc;
      u_long            dr_vers;
      u_long            dr_prog;
      enum status       dr_field;
      struct dupreq     *dr_next;
};
enum progress   {IN_PROGRESS, NO_PROGRESS}
enum status     {OK, ERROR}
```

(그림 17) 서비스 캐싱 정보
(Fig. 17) Service Caching Information

3.6.3 버퍼 캐쉬

커널은 모든 화일 시스템에의 엑세스에 관하여 직접적으로 디스크를 읽고 쓰고 할 수 있지만 디스크의 전송속도가 늦기 때문에 이렇게 하면 시스템의 응답시간과 효율이 나빠진다. 따라서 커널은 버퍼 캐쉬라는 내부 데이터 버퍼 풀(Buffer pool)을 둠으로써 디스크 억세스 빈도를 가능한 한 줄이고 있다. 이 버퍼 캐쉬에는 가장 최근에 사용했던 디스크 블록들의 데이터가 들어 있다.

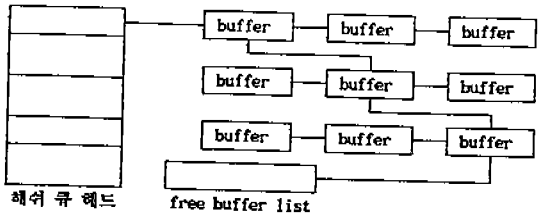
버퍼 헤드(buffer head)에는 디스크상에 있는 데이터가 어떤 화일 시스템 중의 어떤 블록의 것인가를 지정하고 버퍼를 식별하기 위하여 장치 번호 필드(device number feild)와 블록 번호 필드(block number feild)가 있다. 장치 번호는 논리 화일 시스템 번호이며 물리 장치(디스크)번호를 뜻하는 것은 아니다. 버퍼 헤드는 버퍼에 대한 데이터 배열(적어도 디스크 블록과 같은 크기여야 한다)을 나타내는 포인터(pointer)와 버퍼의 현 상태를 나타내는 상태 필드로 갖고 있다. 버퍼의 상태는 다음과 같은 상태의 조합이다. 첫째, 버퍼가 현재 노크되어 있다. ("locked"과 "busy"는 같은 의미이다. "free"와 "unlocked"도 마찬가지이다) 둘째, 버퍼가 유효 데이터를 갖고 있다. 셋째, 커널이 이 버퍼를 재 할당하기 전에 버퍼의 내용을 디스크에 출력해야 한다. 이 상태를 "지연 출력(delayed write)"이라 한다. 넷째, 커널이 현재 버퍼의 내용을 디스크에서 읽어들이든지 디스크에 출력하고 있다. 다섯째 프로세스가 현재 버퍼가 해방되기를 기다리고 있다.

버퍼 헤드는 또한 두 조의 포인터도 갖고 있다. 이들은 다음에 설명하는 바와 같이 버퍼 풀(Buffer pool)의 전체 구조를 관리하기 위하여 버퍼 할당 알고리즘을 사용하였다.

버퍼 캐쉬 시스템에서 데이터를 저장할 공간을 할당할 때는 3가지의 경우를 고려해야 한다. 첫째, 해쉬 테이블이 원하는 데이터가 들어있는 버퍼가 존재할 경우이다. 둘째, 해쉬 테이블에 원하는 데이터가 들어 있는 버퍼가 없을 경우이다. 이때에는 자유 리스트에서 하나의 버퍼를 할당받아 사용한다. 마지막으로 자유 버퍼 리스트에 새로 할당받을 버퍼가 존재하지 않을 때이다. 이 경우에는 현재 사용 중에 있는 버퍼 중에서 가장 오랫 동안 사용하지 않은 버퍼의 내용을 지역 디스크 장치에 대피시키고 그 버퍼를 할당한다.

처음 버퍼 캐쉬가 초기화될 때는 모든 버퍼를 자유 버퍼 리스트에 두고 필요할 때마다 자유 버퍼를 하나씩 할당 받아 사용한다. 또 버퍼 캐쉬에서 사용하는 모든 버퍼는 해쉬 테이블의 인덱스로 참조할 수 있게 해준다. 이렇게 하여 해쉬 함수를 이용하여 해당 버퍼를 쉽게 찾아갈 수 있게 된다.

버퍼 캐쉬는 다음과 같은 구조를 가지게 된다. 각각의 버퍼는 해당 해쉬 큐 헤드에 연결되어 있어 해당 버퍼를 찾을 때는 해쉬 함수 해당 버퍼를 찾아 가게 된다. 또한 사용되지 않은 버퍼는 자연히 자유 버퍼 리스트에 연결된다.



(그림 18) 버퍼 캐쉬의 시스템 구성
(Fig. 18) Buffer cache System Configuration

4. 결 론

본 논문에서는 비대칭형 구조인 Andrew File System, Sprite File System, LOCUS, CODA File System의 특징들을 바탕으로 VDFS(V-Distributed File System)을 설계 및 구현하였으며, 비대칭형 구조가 대형 시스템 위주로 설계되어 있는 것을 소형 컴퓨터에 맞게 설계·구현하였다.

소형 컴퓨터에 알맞는 대칭형 구조는 비대칭형 구조에 비해 전역 접근 열람, 투명성, 시스템 성능 등과 같은 기능 제공 방법에 있어 뒤떨어진다 VDFS 시스템에서는 다음과 같이 서비스를 개선하였다.

첫째, 대칭형 구조의 단점인 전역 접근 열람의 부재와 투명성 제공의 미흡 및 성능 저하 등을 보완하기 위하여 원격 마운트 구조를 기반으로 사용자에게 전역 접근 열람을 제공하였다.

둘째, 접근 투명성, 위치 투명성 및 read-only 중

복 투명성을 제공하여 사용자에게 화일의 접근시 편리성과 데이터의 일관적인 유용성을 제공하였다.

셋째, 캐싱 기법을 사용하여 원격 화일 호출에 대한 서버 부하와 통신량을 감소시키므로써 교착상태 발생을 전제에 두지 않은 상태에서 디스크 없는 워크스테이션에서는 기존의 NFS 보다 원격 서비스 성능을 향상 시켰으며, 디스크가 존재하는 워크스테이션 상에서 화일 접근 속도를 증가시켜 시스템의 성능 향상시켰다.

앞으로 연구 할 방향으로는 본 논문에서 설계된 것을 바탕으로 마이크로 컴퓨터, 퍼스널 컴퓨터, 워크스테이션을 연결하여 화일을 각각의 노드에 분산시켜 완전한 분산 화일 시스템을 구현시키기 위한 교착상태 문제와 상호배제에 관해서 더 많은 연구가 되어져야 할 것으로 사료됩니다.

참 고 문 헌

[1] G. A. Champine, Distributed Computers System, North-Holland, 1980.

[2] P. Wegner, Research Direction in software Technology Distributed processing, The MIT Press, pp. 661-638, 1979.

[3] A. P. Rifkin, M. P. Forbes, R. L. Hamilton, M. Sabrio, S. Shah and K. Yueh, "RFS Architecture Overview", USENIX Summer pp. 248-259, 1986.

[4] R. Sandberg, D. Gddberg, S. Keliman, D. Walsh and B. Lyon, "Design and Implementation of the Sun Network File System", USENIX Summer pp. 119-130, 1985.

[5] D. R Browbrige, L. F. Marshall and B. Randell, "The New Castle connection or UNIXes of the world write", soft. Proc. Exper. 12, pp 1147-1162, 1982.

[6] G.Popek and B. J. Walker, The LOCUS Distributed System Architecture, The Massachusetts Institute of Techinology, 1985.

[7] J. K Ousterhout et. al., "The sprite Network Operating System", IEEE Computer, pp. 23-26, Feb. 1988.

[8] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access", IEEE Computer, May. pp. 9-21, 1990.

[9] Kevin A. Stoodley, E. stewart LEE, "Distributed Disk Caching in a Fileserver/Workstation Environment",

[10] Rashid R. F., "Accent : A Communication Oriented Network Operating System Kernal", Proc. of the English Symposium on Operating System Principles, 1981.

[11] Guilemont Marc, "The Chorus Distributed Operating System : Design and Implementation", International Symposium in Loca Computer Networks, Florence Italy, 1982.

[12] Amnon Barak and Anilitman, "MOS : A Multicomputer Distributed Operat System", Department of Computer Science, The Hebrew UNIX.

[13] D. R. Cheriton "The V kernel : A Software Base for Distributed System", IEEE Software, April 1984.

[14] W. F. Tichy, Z.Rua, Towards a Distributed File System, USENIX Summer, 1984.

[15] Amilitman, "A Distributed Unix System", Bell Communication Research.

[16] K. N. chu and C. chug, "Distributed Computing Systems", Computer Society Vol. 3, No. 1, Jun. 1988.

[17] A. S and R. V. Renessee, "Distributed Operating System", ACM Computing surveys Vol. 17, No. 4, 1985.

[18] 이 병관, 윤 동식, "분산 운영 체제 화일 설계 및 구현 방안 연구", 한국 정보 과학회 추계 학술 논문 발표회, 1993. 10.

[19] 이 병관, 윤 동식, "UNIX 환경하에서 분산화일 설계", 한국 정보 처리 응용 학회 춘계 학술 논문 발표회, 1994. 4.

이 병 관

1979년 부산대학교 졸업(학사)
1986년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1990년 중앙대학교 대학원 전자계산학과 졸업(이학박사)
1988년~현재 관동대학교 전자계산공학과 부교수
관심분야 : 소프트웨어 공학, 분산운영체제.

윤 동 식

1992년 관동대학교 정보처리학과 졸업(학사)
1994년 관동대학교 대학원 전자계산공학과 졸업(공학석사)
1992~94년 관동대학교 전자계산공학과 조교
1994년~현재 원주전문대학, 동우전문대학 삼척산업대학 외래 강사
1995년~현재 아세아항공전문학교 정보통신공학과 교수
관심분야 : 분산운영체제, 멀티미디어, 객체지향 언어.