□특집□

고가용성 리눅스

최 재 영† 최 종 명†† 김 은 희†† 김 민 석††

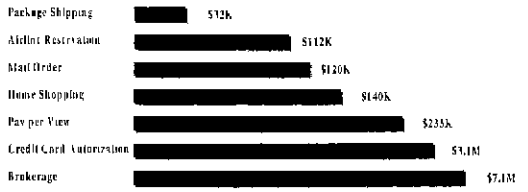
◈ 목 차 ◈

1. 서 론

오늘날 전세계적인 마케팅시장에서 금융서비스, 원격통신, 의학에서부터 소매, 수송, 제조에 이르는 많은 종류의 기업체들은 하루 24시간, 1년 365일 동안 사업에 필요한 정보를 항상 서비스해 주고 있다. 이런 기업체들은 고객의 요구에 대한 서비스를, 예를 들면 은행 잔고 조회, 투자 기금 기록, 테스트 결과, 여행 예약 정보 등과 같은 중요한 정보를 정확하고 지속적으로 제공해 주어야 한다. 이런 서비스를 할 수 있는 관건은 기업체의 정보 기술의 가용성(availability)에 있다고 말할 수 있다.

고가용성(high availability)이란 하드웨어, 소프트웨어, 네트워크 등에 문제가 발생하였을 때 지속적으로 서비스를 제공해주는 기술을 말한다. 고가용성은 정보화 시대의 도래, 전자 상거래의 발전 및 정보 및 멀티미디어 서비스 등의 컴퓨팅 환경의 변화에 따라 점차 중요시되고 있다. 특히 하루 24시간 일년 내내 서비스를 제공해주어야 하는 네트워크 컴퓨팅 부분에서 고가용성은 필수 불가결한 것으로 인식되고 있다.

Contingency Planning Research에서 제공하는 연구에 의하면, 평균적인 시스템의 중단으로 기업체들은 몇 백만 달러의 손해를 입는다고 나와 있다. 더욱이 중요한 것은 예상치 못한 시스템의 이상으로 정보를 잃어버린 기업체들의 50%는 다시 서비스를 제공할 수 없으며, 이런 경우 90%는 2년 안에 도산하는 것으로 밝혀졌다 [1]. 1995년에 Oracle과 Datamation의 보고서에 의하면 갑작스런 서비스 중단은 평균적으로 기업체에 시간당 80,000~350,000 달러의 손실을 입히는 것으로 나타났다 [4]. 또한 포춘 500대 기업은 해마다 4시간씩 9차례나 서버가 예상치 못하게 정지되는 것을 경험했다. 경제적 손실은 서비스가 중단될 때마다 $100,000에서 $500,000 규모였으며 연간 3백만 달러의 손실을 보았다. 이러한 손실 금액은 잠재적인 이윤과 손실을 고려하지 않은 금액이다 [3]. 이러한 사례에서 볼 수 있듯이 고가용성은 정보화 시대의 모든 기업들에 꼭 필요한 요소이다. 그림 1은 갑작스런 서비스 중단에 따른 시간당 손실 비용을 서비스 업체별로 보여주고 있다.



(그림 1) 갑작스런 서비스 중단으로 인한 업체별 시간당 비용 손실

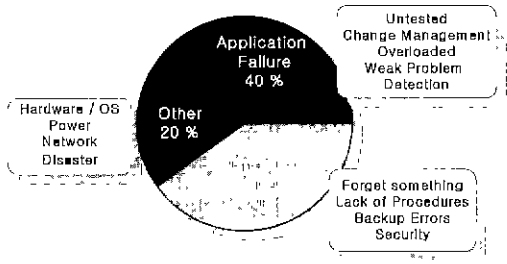
† 정회원 : 숭실대학교 컴퓨터학부 조교수
†† 정회원 : 숭실대학교 대학원 박사과정

2. 서비스 중단의 요인과 SPOF

서비스 중단은 계획된 중단(planned down)과 갑작스런 중단(unplanned down)으로 구분할 수 있다.

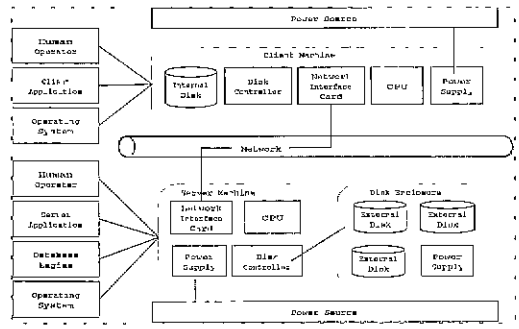
- 계획된 중단 : 백업, 업그레드, 유지, 보수 등 여러 가지 계획에 의해 서비스가 중단되는 경우.
- 갑작스런 중단: 정전, 하드웨어 및 소프트웨어적 오류, 해킹, 사용자 오류, 자연적 재해 등의 원인으로 서비스가 중단되는 경우.

1998년 가트너 그룹의 조사에 의하면 갑작스런 서비스 중단의 원인은 운영자의 실수나 응용 프로그램의 실패로 인한 것이 80%에 달하는 것으로 알려졌다 [5]. 따라서, 고가용성 시스템을 사용하면서 운영자의 교육 및 프로세스의 자동화가 필수적으로 수반되어야 한다.



(그림 2) 갑작스런 서비스 중단의 원인

한 요소의 이상으로 전체 서비스가 중단되는, 하나의 소프트웨어 또는 하드웨어 요소를 SPOF (Single Point Of Failure)라고 한다 [2]. 하드웨어적인 문제에 의한 서비스 중단은 SPOF에 해당되는 요소들을 중복하여 설치하므로서 해결할 수 있다. 이때 시스템에서 장애가 발생할 수 있는 SPOF 요소들을 찾아내는 것이 중요하다. 그림 3은 시스템이 가지는 일반적인 SPOF 요소들이다. 그림에서 볼 수 있듯이 CPU, 네트워크 카드, 네트워드 등은 모두 SPOF가 될 수 있다 [2].



(그림 3) SPOF 요소들

3. 리눅스 고가용성

Sun Microsystems, Compaq, SGI, Hewlett Packard 등의 회사들의 리눅스 지원과 국내 리눅스 전문 회사의 탄생으로 볼 때, 리눅스 시장은 앞으로 상당히 활성화될 것이다. 예상되는 리눅스의 시장의 경제적 전망을 살펴보면 표 1과 같다.

(표 1) 리눅스 시장성

(단위: 백만 달러)

제품 \ 연도	1999년	2000년	2001년
Linux Server Revenue	26.9	41.2	53.1
Linux Server Shipments	3,894	5,105	6,738

리눅스는 유닉스, 윈도우 NT에 비해 고가용성 시스템 개발에 더 유리한 입장에 있다. 즉, 리눅스는 다른 시스템에 비해 다음과 같은 장점들을 가지고 있다.

- 공개 소프트웨어
- 저렴한 가격
- 고성능

• 유닉스와 유사

리눅스 고가용성 시스템 개발을 위한 Linux-HA [7]라는 공개 프로젝트는 고사용성 시스템, 고가용성을 위한 파일 시스템, 모니터링 시스템 개발 등 여러 방면에서 솔루션들을 제공하고 있다. 현재 Linux-HA 프로젝트는 Heartbeat [8], Fake [9], Heart [10] 등의 고가용성 시스템을 위한 프로젝트와 High Availability RAID [11], GFS [12], LFS [13], CODA [14] 등의 파일 시스템, Mon [15], PIKT [16], NOCOL/SNIPS [6] 등의 모니터링 시스템 등 다양한 분야와 밀접하게 연계되어 진행되고 있다. 그 중에서 Heartbeat은 시리얼, UPD, PPP/UDP 미디어를 통한 노드 체크 기능과 IP 주소 takeover 기능, 리소스 그룹을 포함한 뛰어난 리소스 모델 등을 지원하면서, 실제 서비스에 활용하고 있다. 또한 Linux-HA 프로젝트 웹사이트는 하루 1000회 이상의 방문이 이어지고, 점차적으로 이 프로젝트에 참여하는 지원자들이 늘고 있다.

Linux-HA 공개 프로젝트 중 대표적인 Fake, Heartbeat, Mon에 대해 자세히 알아본다.

3.1 Fake

Fake는 부가적인 인터페이스나 ARP 스푸핑(spoofing, 속임수)을 사용하여 동일 네트워크에 있는 다른 기계의 IP 주소를 넘겨받을 수 있다. 부가적인 인터페이스는 물리적인 인터페이스이거나 IP 별칭일 수 있다.

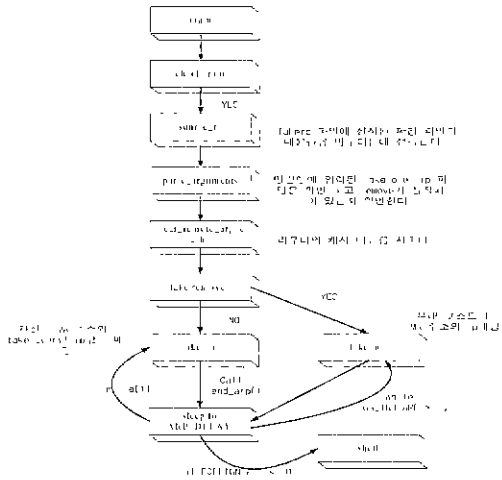
그림 4와 같이 주 서버가 IP 주소 192.168.20.11로 서비스를 하고 있다가, 주 서버가 갑작스러운 문제로 인하여 서비스를 제공하지 못할 때 대기중인(standby) 서버는 주 서버의 IP 주소 192.168.20.11을 받아서 서비스를 수행하게 된다. 이렇게 IP 주소를 넘겨받는 것이 Fake의 기본적인 기능이다.

interface	Node 1 (primary)	Node 2 (secondary)
service	192.168.20.11	192.168.20.20
standby	192.168.21.10	192.168.21.20
interface	Node 1 (primary)	Node 2 (secondary)
service		192.168.20.20
standby	192.168.20.11	192.168.21.20

(그림 4) Fake의 IP takeover

Fake는 그림 5과 같은 형태로 동작하게된다. Fake를 수행하기 위해 .fakerc와 clear_routers, 대상 IP 구성 파일을 읽어서 환경변수들을 설정한다. Fake는 ARP 프로토콜을 이용해 IP를 스푸핑한다. Fake는 다음과 같은 순서로 실행된다.

- 사용자가 root인지 확인한다.
- Fake 환경변수 설정한다.
- 라우터의 캐시에서 해당 IP의 내용을 삭제한다.
- Fake할 IP의 논리적 네트워크 인터페이스를 설정한다.
- 서버의 라우팅 테이블에 등록한다.
- 주기적으로 자신의 MAC 주소와 Fake IP 정보를 가진 ARP 패킷을 브로드캐스팅한다.

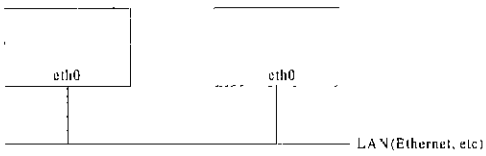


(그림 5) Fake 수행 모습

3.2 Heartbeat

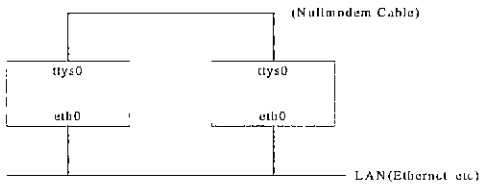
Heartbeat은 Alan Robertson이 공개적으로 진행 중인 리눅스 고가용성 프로젝트로서 기본적인 시스템 점검 기능과 IP takeover 기능을 가지고 있다. Heartbeat 메시지는 UDP, Serial, PPP/UDP 등 다양한 미디어들을 통해서 전송될 수 있다. 만약 메시지가 전송되지 않으면, timeout 카운트가 시작되고, 계속해서 얼마동안 Heartbeat 메시지를 받지 못하면, 노드는 장애가 발생한 것으로 간주한다. 살아있는 다른 노드들을 크로스 체킹(cross checking)하여 다른 노드들이 서로 동의할(voting) 경우에 장애로 판단된다.

Heartbeat을 위한 기본적인 하드웨어적인 구성은 그림 6과 같다. 이러한 경우에는 TCP를 통해서 두 개의 노드가 Heartbeat 메시지를 주고받을 수 있다.



(그림 6) 간단한 구성

그러나, 이러한 구성은 네트워크 카드에 문제가 발생하는 경우에 작동하지 않을 수 있기 때문에 일반적으로 그림 7과 같은 방법으로 구성한다. 그림 7은 네트워크 카드에 문제가 발생할 수 있는 경우를 고려해서 serial 라인을 통해서 Heartbeat 메시지를 서로 주고받을 수 있다.



(그림 7) 시리얼 라인을 이용한 구성

현재 클러스터에 포함된 노드들 사이에 주고받는 Heartbeat의 메시지는 다음과 같이 (name, value)의 쌍으로 되어 있다.

```
struct ha_msg {
        int    nfields;  /* 메시지의 필드 수 */
        int    nalloc;   /* 최대 가질 수 있는 필
                            드의 수 */
        char ** names;   /* 필드 이름 */
        char ** values;  /* 필드의 값 */
        };
```

각 노드들은 Heartbeat 메시지에 포함된 필드와 그 값을 보고 자신과 다른 노드의 현재 상태를 파악한다. 메시지에 포함되어 있는 필드의 이름으로는 다음과 같은 것이 정의되어 있다.

- F_TYPE - 메시지 타입
- F_ORIG - 출발지
- F_TO - 목적지
- F_STATUS - 새로운 상태
- F_TIME - 타임 스탬프
- F_SEQ - 시퀀스 번호
- F_LOAD - 평균 로드
- F_COMMENT - 주석
- F_TTL - Time To Live

Heartbeat은 두 개의 구성 파일(ha.cf, ipresource)을 가지고 있다. ha.cf는 Heartbeat의 전체적인 구성을 나타내는 파일이다. 다음은 ha.cf 구성 파일에서 사용되는 필드들이다.

- serial - Heartbeat의 시리얼 라인의 구성을 알려준다. 하나 이상의 시리얼 포트가 있으면 ring 형태임을 의미한다.
- node - 클러스터에 포함된 노드들을 의미한다.
- keepalive - Heartbeat 메시지가 전송되는 시간 간격

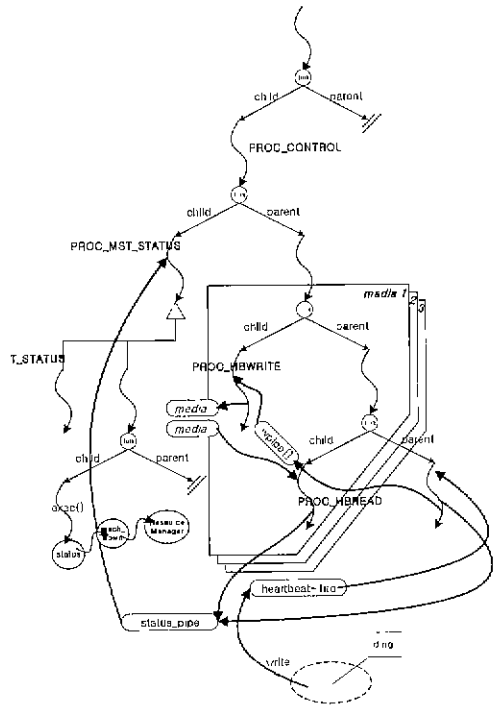
- deadtime - 노드가 죽었다고 선언하기 전에 기다리는 시간
- hopfudge - 메시지를 버리기 전까지 커질 수 있는 hopcount

ipresources 파일은 Heartbeat에서 관리하는 자원들에 관한 정보를 기록한다. Heartbeat은 고가용성 시스템에서 서비스하는 내용을 자원으로 표현한다. 자원은 IP 주소, NIC, 파일 시스템, 디스크, 응용 프로그램 등을 의미한다. 자원들은 서비스하는 서버가 다운될 때 자원 관리자(Resource Manager)를 통해 다른 서버로 전달된다.

Heartbeat은 데몬 형태로 실행되며, 여러 개의 자식 프로세스들을 생성한다. 하나의 미디어를 사용하는 경우에 기본적으로 아래와 같은 4개의 프로세스가 생성되고, 미디어가 하나씩 증가될 때마다 두 개의 프로세스(PROC_HBREAD, PROC_HBWRITE)가 더 생성된다.

- Control process(PROC_CONTROL) - 자식 프로세스들을 시작시키고, 클러스터에 전송할 명령어를 FIFO에서 읽는다.
- Status process(PROC_MST_STATUS) - status pipe를 읽고, 자식 프로세스들을 생성해서 상태를 변경하는 작업을 수행시킨다. 또한 주기적으로 keepalive 메시지를 전송한다.
- hb channel read process(PROC_HBREAD) - Heartbeat 채널마다 하나씩 생성되는 이 프로세스는 Heartbeat 채널을 읽어서, 메시지를 status pipe에 복사한다.
- hb channel write process(PROC_HBWRITE) - Heartbeat 채널마다 하나씩 생성되는 이 프로세스는 자신의 pipe를 읽어서 결과를 미디어에 전송한다.

Heartbeat는 전체적으로 아래 그림 8과 같은 형태로 수행된다. 그림에서 굵은 선은 자료의 흐름을 나타내고, 얇은 선은 제어의 흐름을 의미한다. ding() 함수는 일정 시간 간격으로 Heartbeat 메시지를 작성해서 heartbeat-fifo에 기록한다. PROC_CONTROL 프로세스는 heartbeat-fifo에서 값을 읽어서, status_pipe와 미디어의 wpipe에 기록한다. PROC_HBWRITE는 미디어의 wpipe에서 메시지를 읽어서, 미디어를 통해 클러스터에 속해있는 다른 노드에 메시지를 전달한다. 다른 노드에서도 동일한 방법으로 PROC_HBREAD 프로세스에서 미디어를 통해 전달된 메시지를 읽는다. 이 메시지는 status_pipe에 기록되고, 이것은 STATUS_PROC 프로세스가 읽고, 노드의 상태를 파악해서 적절한 작업을 수행한다. 만약, 서비스하는 노드가 정지된 경우에는 프로세스를 생성해서 status라는 쉘 프로그램을 수행한다. status 쉘 프로그램은 mach_down과 ResourceManager 프로그램을 이용해서 자원들을 넘겨받는다.



(그림 8) Heartbeat 수행 모습

3.3 Mon

Mon은 서버에서 제공하는 서비스의 가용성을 모니터링 하는 기능을 한다. Mon은 크게 두 가지 작업을 수행한다. 하나는 서비스의 상태를 모니터링하는 것이고 다른 하나는 서비스가 중단되었을 때 적절한 조치를 취하는 것이다. 이러한 기능은 Mon 스케줄러에서 수행하며, Mon 스케줄러는 계속적으로 각 서비스들의 상태를 모니터링하고 서비스 중단이 발견될 경우에 적절한 경고 작업(alert)을 호출한다.

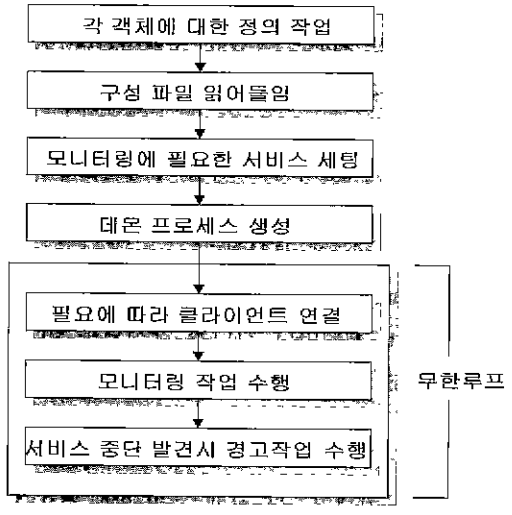
Mon은 다음과 같은 요소들로 구성되어 있다.

- Monitors : Monitors는 서비스의 상태를 체크하고, 정해진 출력장치를 통해 서비스의 상태를 보고하는 프로그램이다. Mon 스케줄러와는 독립적 작동하기 때문에 새로운 서비스 테스트를 추가하려면, 해당되는 모니터 스크립트를 작성해서 적당한 디렉토리에 삽입해 주면 동작한다.
- Alerts : Alerts는 Mon이 서비스 중단을 감지했을 때, 경고 메시지를 보낸다거나, 그 밖의 다른 행동을 취하는 스크립트이다. Monitors와 마찬가지로 Mon 스케줄러에 대해 독립적이기 때문에 적당한 스크립트를 쉽게 추가할 수 있다.

Mon은 클라이언트/서버 형태로 구성되어 작동하며, 클라이언트와 서버는 각각 다음과 같은 역할을 수행한다.

- Server : Mon 서버는 서비스 모니터링과 경고 작업이 행해지는 부분이다. Mon 스케줄러가 시작되면, 모니터링에 필요한 서비스들을 결정하기 위해 환경 파일을 읽어 들인다. 그 후 스케줄러는 무한루프를 돌면서 클라이언트 연결, 모니터링 작업, 그리고 서비스 중단에 대한 경고 작업을 수행한다. 각 서비스들은 구성 파일에 설정되어 있는 정해진 시간 간격을 통해서 점검된다.
- Client : Mon 클라이언트는 Mon 스케줄러에 특정 명령을 내려서 원하는 정보를 얻을 수 있다. 예를 들면 특정 호스트나 호스트 그룹, 서비스들을 작동시키거나 멈출 수 있고, 서비스의 상태를 저장하거나 불러올 수 있으며, 서버를 초기화시키거나(reset) 서버에 대한 인증 작업을 명령할 수 있다.

Mon 스케줄러는 그림 9와 같은 형태로 작동하게 된다. Mon이 실행되면 일단 구성 파일을 읽어 들인다. 구성 파일에는 글로벌 옵션, 호스트 그룹, 서버그룹 등이 정의되어 있으며, 각 서비스별 모니터링 및 경고 작업의 수행을 위한 정보들이 저장되어 있다. Mon 스케줄러는 구성 파일의 정보를 기반으로 각 서비스에 대한 모니터링 작업을 세팅한다. 모니터링 작업을 위한 준비 작업이 끝나고 나면, 프로세스를 생성하고 데몬 프로세스로 만든다. Mon 데몬은 루프를 돌며 계속적으로 모니터링 작업을 수행하고, 서비스 중단이 발견되면 경고 작업을 수행하게 된다.



(그림 9) Mon 수행 과정

4. 결 론

오늘날 고가용성 문제는 서버 시장에서의 가장 중요한 문제라고 할 수 있다. 인터넷 서비스가 활성화될수록, 갑작스러운 서비스 중단은 경제적으로 막대한 손실을 가져올 수 있다. 어쩔 수 없는 서비스 중단 상황에서 이러한 경제적인 손실을 줄이기 위해서 고가용성 시스템에 대한 연구와 개발은 매우 중요하다.

국내외적인 상황과 리눅스 시장의 활성화, 서버의 운영체제로서의 리눅스의 장점들로 인해 리눅스 플랫폼의 인터넷 서비스가 증가하고 있다. 그에 따라 리눅스 고가용성 시스템의 개발에 대한 요구가 커지고 있다. 이러한 요구에 따라 Linux-HA라는 공개 프로젝트가 진행 중에 있다. 이 프로젝트에서는 여러 분야에서 다양하고 활발하게 진행 중에 있지만, 다음과 같은 문제점들이 있다. 첫째, 공개 프로젝트로 개발된 소프트웨어들은 기능이 부족하다. 공개 소프트웨어들이 대부분 테스트를 위한 프로토타입 개발을 목표로 하기 때문에, 상용 소프트웨어와 비교하여 기능면에서 많이 뒤떨어져 있다. 둘째, 개발된 소프트웨어들을 통합하여 사용하기 어렵다. 각 소프트웨어들은 개별적으로 개발되어 있기 때문에 통합되어 사용하기 위해서는 많은 수정이 필요하게 된다. 세번째 문제점으로는 개발된 소프트웨어들이 독립된 언어로 개발되어 있다는 것이다. 일부는 C 언어로 개발되어 있고, 또 다른 소프트웨어는 Perl이나 Tcl/Tk 등과 같은 언어로 작성되어 있기 때문에 통합해서 사용하기에 어려움이 많다. 마지막 문제점으로는 공개용으로 개발된 소프트웨어들은 사용자의 편의성을 전혀 제공하지 않는다는 것이다. 그러나, 이러한 문제점들은 앞으로 프로젝트가 진행되면서 점차 해결될 것으로 기대된다.

참 고 문 헌

[1] About Geographic High Availability, http://www.clam.com/clam/whatisgeoha.html

[2] Peter S. Weygant, Clusters for High Availability, Hewlett-Packard Press, 1996.

[3] HP High Availability Service Provide Competitive Advantage, http://www.hp.com/ibpprogs/csy/advisor/mar97/support/cover.html

[4] Linux High Availability HOWTO, http://metalab.unc.edu/pub/Linux/ALPHA/linux-ha/High-Availability-HOWTO.html

[5] High Availability Solution, http://www.unisys.com/marketplace/ent/ha-nt/ha-applicationdowntime-zoom.html

[6] nocol, http://www.netplex-tech.com/software/nocol/

[7] High-availability Linux Project, http://linux-ha.org/

[8] Heartbeat, http://www.henge.com/~alanr/ha/download/

[9] Fake: Redundant Server Switch, http://linux.zipworld.com.au/fake/

[10] Heart a redundant, distributed cluster technonogy, http://www.lemuria.org/Heart/

[11] Software-RAID HOWTO, http://www.linas.org/linux/Software-RAID/Software-RAID.html

[12] The Global File System, http://gfs.lcse.umn.edu/

[13] The Logging Filesystem, http://hp.cso.uiuc.edu/~c-cook/prof/lfs

[14] Coda File System, http://www.coda.cs.cmu.edu/

[15] MON, Service Monitoring Deamon, http://www.kernel.org/software/mon/

[16] PIKT, http://pikt.uchicago.edu/pikt/

최 재 영

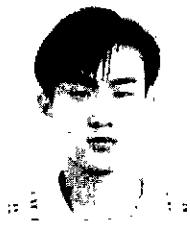
1984년 서울대학교 제어계측공학과 (학사)
1986년 미국 남가주대학교 전기공학과(컴퓨터공학) (석사)
1991년 미국 코넬대학교 전기공학부 (컴퓨터공학) (박사)
1992년-1994년 미국 국립 오크리지연구소 연구원
1994년-1995년 미국 테네시 주립대학교 연구교수
1995-현재 숭실대학교 정보과학대학 컴퓨터학부 조교수
관심분야 : 시스템 소프트웨어, 병렬/분산처리

김 은 희

1993년 숭실대학교 전자계산학과 (학사)
1998년 숭실대학교 대학원 전자계산학과 (석사)
1998년-현재 숭실대학교 대학원 컴퓨터학과 박사과정
관심분야 : 분산시스템, 병렬처리

최 종 명

1992년 숭실대학교 전자계산학과 (학사)
1996년 숭실대학교 대학원 전자계산학과 (석사)
1997-현재 숭실대학교 대학원 컴퓨터학과 박사과정
관심분야 : CSCW, 시각프로그래밍, 분산처리



김 민 석

1997년 숭실대학교 소프트웨어 공학과 (학사)
1999년 숭실대학교 대학원 전자계산학과 (석사)
1999-현재 숭실대학교 대학원 전자계산학과 박사과정
관심분야 : 병렬/분산컴퓨팅, 시스템 소프트웨어, 병렬 알고리즘