

# GNU/Linux를 이용한 4-node cluster 제작

최기영 xenus@hlug.hanyang.ac.kr

2002년 3월 24일

## 요약

이 문서는 HLUG<sup>1</sup> 내부 세미나용으로 작성되었으며 GNU/Linux 를 채용한 4대의 PC를 이용해 수치계산용 4-node cluster를 구축하는 과정을 따라가고 있다. 네트워크나 시스템의 속도를 위한 최적화 과정은 생략되어 있으며 Lam 을 이용해서 시스템들을 메인노드에서 인식시키는 과정까지만 설명되어 있다. 보다 전문적인 내용과 세밀한 설정은 다른 문서를 참조하기 바란다.

Copyright (c) 2002, 최기영  
This document released under GFDL.

---

<sup>1</sup><http://hlug.hanyang.ac.kr>

# 제 1 절 Introduction

## 1.1 What is a cluster?

클러스터(cluster)라고 불리우는 시스템은 간단히 말해서 하나의 컴퓨터처럼 동작하는 여러개의 단일 컴퓨터들의 집합체라고 할 수 있다. 약간의 시스템과 network상의 설정 그리고 프로그램의 작성에 있어서 LAM/MPI(혹은 PVM)를 이용한 코딩이 필요하긴 하지만 여러개의 단일 컴퓨터들을 묶음으로써 하나의 고성능 시스템을 얻을 수 있다는 점에서 클러스터링(clustering)은 대단히 큰 매력과 장점을 지니고 있다고 할 수 있다.

일반적으로 수Giga byte이상의 메모리와 현재 일반화되어 있는 CPU보다 훨씬 빠른 CPU를 갖추고 있는 고성능 시스템을 구입하는데는 안타깝게도 많은 비용을 지불해야 한다. 하지만 클러스터를 이용하게 되면 굉장히 저렴한 비용으로 이와 동일한 효과를 얻을 수 있다. 예를 들어, 이 문서에서 작성한 것과 같이 2기가의 메모리와 4개의 Athlon XP 1800+ CPU를 갖춘 클러스터와 동일한 성능의 워크스테이션(Workstation)을 구입하기 위해서는 수천만원의 비용을 들여야 하지만 실제로 이 클러스터를 꾸미는데 든 돈은 3백만원이 조금 안되는 비용이었다.

## 1.2 COW and Beowulf

여기서 우리는 Cluster of Workstation과 Beowulf가 어떻게 다르며 Beowulf는 무엇인지에 대해 확인해볼 필요가 있다. COW는 말 그대로 완벽하게 독자적으로 동작 가능한 Workstation(혹은 PC)들을 하나의 클러스터로 묶은 시스템을 말한다. 각각의 노드(node)<sup>2</sup>들은 독자적으로 사용이 가능하며 운영체제들도 모두 설치되어 있다. 굳이 클러스터로 동작하지 않더라도 별도로 프로그램을 실행하고 사용될 수 있기 때문에 클러스터로써 사용되지 않을 때에는 각 시스템들을 다른 사용자들이 각자의 목적에 의해 다른 용도로 사용할 수 있다. 그렇지만 기본적으로 모든 시스템들이 하드 디스크와 비디오 카드를 비롯한 장치들을 갖춰야 하기 때문에 전체적인 구축 비용이 늘어난다. 만일 상용 운영체제를 사용하려 한다면 노드수 만큼의 운영체제 구입 비용을 추가해야 할 것이다.

반면에 Beowulf는 이와는 조금 다른 특징을 지니고 있다. beowulf는 하드 디스크, 비디오 카드 등을 갖추고 있지 않으며 오로지 CPU, mother board, ram등만을 갖추고 있으며 외부에서 바라봤을때 하나의 시스템으로 보인다. beowulf는 한마디로 병렬처리에 최적화 되어 있는 시스템이며 COW에 비해 훨씬 저렴한 구축비용이 든다. beowulf에서 각 노드를 별도로 사용할 수 있는 방법은 없으며 접근역시 메인노드를 통해서만 가능하다.

그렇다면 어느쪽이 더 좋은가? 답은 자신이 과연 어떤 용도로 클러스터를 꾸미려 하는지에 있다. 병렬계산 이외에 다른 용도로 쓸 일이 없다면 beowulf는 그 해답이 될 것이며 약간의 하드웨어 비용을 더 지불하더라도 각 노드들을 평소에 다른 용도로 쓰며 필요한 경우에만 병렬 시스템으로 이용하려 한다면 COW가 그 해답이 될 것이다.

내가 이제부터 구축하려는 4-node짜리 클러스터(MINION이라는 이름을 갖고 있다.)는 전형적인 COW로 자원소모가 많은 수치계산을 할 때에는 각 노드를 모두 사용하지 않지만 그렇지 않은 가벼운 계산이나 기타 목적으로 쓸 때는 연구실의 구성원들이 현재 비어있는 노드에서 자신의 작업을 할 수 있게 구성되어 있다.

## 1.3 하드웨어 구입

MINION을 구축하는데 쓰인 하드웨어는 다음과 같다.

- Athlon XP 1800+ CPU (4EA)
- VIA KT133A Mother Board (4EA)
- GeForce2 Graphic Card (4EA)
- 3Com 3CSOHO100-TX Lan Card (5EA)

<sup>2</sup>클러스터에 묶여 있는 각각의 컴퓨터들을 노드 라고 부른다

3Com 3C16790 dual speed switch 5 plus hub (1EA)

IMT4000 System Case (4EA)

Samsung 3.5 inch Floppy disk drive (4EA)

모든 부품은 용산전자상가에서 구입했으며 가격 변동이 심한 컴퓨터 부품의 특성상 별개 부품의 가격을 이야기 하는 것은 무의미하나 총 구입 비용은 300만원에 조금 못미쳤다.

클러스터를 꾸미기 위해 하드웨어를 구입할때 가장 중요한 사항은 각 시스템들이 동일한 사양의 하드웨어를 갖춰야 한다는 것과 빠른 network이 지원되어야 한다는 것이다. 만일 노드별 계산 속도가 차이가 나면 전체 성능은 가장 느린 노드에 맞춰지기 때문이며 계산 중간중간 공유해야하는 데이터들을 network을 통해서 주고받기 때문에 network이 느릴경우 빠른 CPU를 구입한 의미가 사라지기 때문이다. 또한 나는 비용을 300만원에 맞추기 위해 그렇게 하지 못했지만 메인노드는 dual system으로 구축하는 것이 좋다. 이유인즉, 메인노드는 외부와 연결된 시스템이며 전체 노드들을 관리하게 된다. 또한 소스의 컴파일과 기타 작업역시 주로 메인노드에서 이루어 진다. 만일 CPU가 2개인 SMP 시스템이라면 하나의 CPU를 이런 작업들을 위해 사용하고 나머지 하나는 계산에 사용할 수 있어서 전체적인 균형을 해치지 않고 작업을 할 수 있다.

## 제 2 절 System configuration

### 2.1 하드웨어 구성 및 운영체제 설치

클러스터 제작에 사용할 수 있는 운영체제의 종류는 제법 많다. 심지어 마이크로소프트의 Windows NT 계열의 운영체제를 이용해서도 할 수 있다. 하지만 그에 따른 막대한 비용을 들여야 한다. 여기서는 운영체제로 Debian GNU/Linux를 사용했다. GNU/Linux의 설치에 대해서 자세히 설명하는 것은 이 문서의 범주를 넘어 서므로 생략하겠다.

각 노드는 Debian GNU/Linux의 base system을 설치했으며 설치 Method는 network을 이용했다. 사용 가능한 할당된 IP가 하나였기 때문에 각 노드에 운영체제를 설치할때마다 랜선을 옮겨 꼽으면서 동일 IP로 설정해 설치한 후 IP정보를 변경했다. 이러한 번거로움이 싫다면 CD-ROM을 연결해서 CD로 설치하는 것도 좋은 선택일 것이다. 난 CD-ROM을 달아보아야 그다지 쓸일이 없다는 판단에 빠버리고 플로피를 이용해서 커널과 하드웨어 모듈만 설치한 후에 나머지는 network을 통해서 설치했다.

메인 노드에 랜카드를 두개 설치해야 한다는 사실을 제외하면 각 노드의 하드웨어 구성은 모두 동일하다.

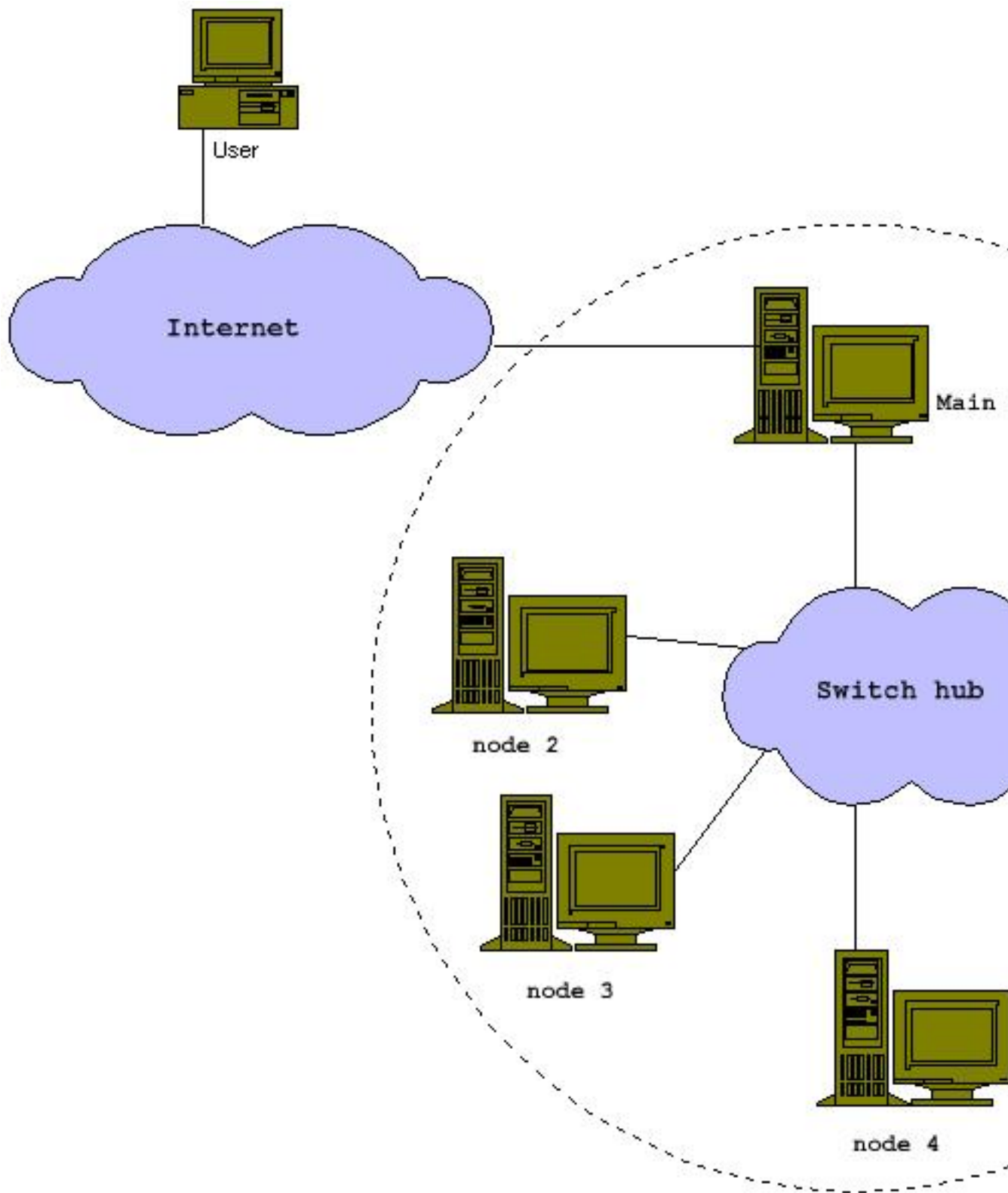
### 2.2 Network 구성도

### 2.3 Network configuration

클러스터를 구축하기 위해 network를 구성하는 것은 일반적인 subnetwork를 구성하는 것과 크게 다르지 않다. 앞절의 구성도에서 이미 확인했듯이 각 노드에는 편의상 node number 라는 이름을 할당했으며 node1 이 메인노드로 외부 network과 연결된 유일한 노드이다. 또한 node1 은 hub와 연결되어 있기도 하다. 나머지 node2, node3, node4는 이 hub에 연결되어 있으며 사설 IP를 할당했다.

우선 각 노드의 네트워크를 통한 연결이 가능하게 해야 했기 때문에 나는 다음과 같이 사설 IP를 할당했다. /etc/hosts 파일을 통해서 사설IP설정을 해주면 된다. 이 파일의 내용은 각 node마다 동일해야 한다.

```
# /etc/hosts
127.0.0.1      localhost
192.168.0.1   node1
192.168.0.2   node2
```



```
192.168.0.3    node3
192.168.0.4    node4
```

또한 node1 은 166.104.53.194 라는 외부에서 접속 가능한 IP 주소도 함께 할당되어 있다. node1의 /etc/network/interfaces 파일을 다음과 같이 작성해 주면 된다.

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# The first network card - this entry was created during the Debian installation
# (network, broadcast and gateway are optional)
auto eth0 eth1
iface eth0 inet static
    address 166.104.53.194
    netmask 255.255.255.0
    network 166.104.53.0
    broadcast 166.104.53.255
    gateway 166.104.53.1
iface eth1 inet static
    address 192.168.0.1
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.169.0.255
    gateway 166.104.53.194
```

node2의 /etc/network/interfaces 파일의 내용은 다음과 같다.

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# The first network card - this entry was created during the Debian installation
# (network, broadcast and gateway are optional)
auto eth0
iface eth0 inet static
    address 192.168.0.2
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
```

node3과 node4의 network 설정도 크게 다르지 않다. address 부분만 할당된 사실IP로 변경해 주면 된다. 이후 network interface를 재시작 시키거나 시스템을 재부팅 시켜서 각 노드들간에 ping이 도달하면 된다.

node1에서 다른 노드들을 인식하지 못한다면, /etc/nsswitch.conf 파일의 내용 중

```
hosts:          files dns
```

이라는 내용이 들어 있는지 확인해보기 바란다. files 가 없으면 /etc/hosts 파일을 참조하지 않고 바로 dns로 쿼리를 보내기 때문에 사실IP를 적용한 node2부터 node4 까지는 인식이 되지 않는다.

## 2.4 NFS configuration

클러스터 시스템에서 사용자는 각 노드들에 모두 접근이 가능해야 한다. 이는 각 노드마다 계정이 있어야 한다는 것을 의미하며 어디서 작업을 했든 결과를 다른 노드에서도 확인이 가능해야 한다는 것 역시 의미<sup>3</sup>한다. 또한 각 노드들은 동일한 버전의 프로그램과 라이브러리를 사용해야 한다. 만일 각 노드들이 별도의 /usr 파티션을 운영한다면 패키지 업데이트를 해줄때도 모두 다 해줘야 한다. 만일 node1의 /usr 파티션을 나머지 노드들이 공유한다면 패키지 업데이트나 버전 관리등을 node1만 해주면 되므로 이 역시 훨씬 관리작업을 수월하게 해준다. 이러한 이유로 클러스터 시스템에서는 NFS를 도입하는 것이 좋다.

이를 위해서 나는 두개의 패키지를 추가 설치했다. nfs-common과 nfs-kernel-server 을 설치했는데 주의할 것은 nfs-user-server를 설치하지 말라는 것이다. User space에서의 nfs가 좀더 많은 기능을 갖고 있긴 하지만 속도가 더 느리다. 클러스터에서 가장 중요한 것이 network의 속도라는 점을 고려할때 선택에서 제외할 충분한 이유가 된다. 데비안에서는 dselect에서 이 두 패키지를 선택해 주는 것으로 모든 설치가 끝난다. 레드햇이나 다른 배포판을 설치했다면 해당 배포판에 맞는 패키지 설치 문서를 확인해보길 바란다.

node1이 nfs server로 동작하며 node2,node3,node4는 nfs client로 동작한다. 이를 위해서 node1의 /etc/exports 파일을 다음과 같이 편집하자.

```
# /etc/exports: the access control list for filesystems which may be exported
#                to NFS clients.  See exports(5).

/home           node2(rw)
/usr            node2(rw)
#/cdrom         node2(ro,all_squash,anonuid=150,anongid=100)

/home           node3(rw)
/usr            node3(rw)
#/cdrom         node3(ro,all_squash,anonuid=150,anongid=100)

/home           node4(rw)
/usr            node4(rw)
#/cdrom         node4(ro,all_squash,anonuid=150,anongid=100)
```

이렇게 해주면 /home 과 /usr 디렉토리는 node2,node3,node4에서 nfs 로 마운트 해서 사용할 수 있다. cdrom에 대한 줄은 모두 주석으로 막혀 있는데 만일 cdrom이 node1에 설치되어 있고 다른 노드들에서 node1의 cdrom을 사용하고 싶다면 주석을 풀고 nfs server를 재시작 하기만 하면 된다.

node2와 나머지 노드들의 설정은 동일하다. node1의 /home, /usr 디렉토리를 마운트 해서 사용하도록 설정하면 된다. 아직 각 노드에 일반 사용자 계정은 추가 하지 않은 상태이므로 /home 파티션은 비어 있을 것이다. 그렇지 않다면 아무것도 없이 깨끗하게 지운 후 디렉토리를 다시 만들기 바란다. /usr 파티션도 마찬가지로 이지만 /usr 은 운영체제를 base만 설치했다고 해도 기본적으로 설치되는 것들이 있으므로 비어있지는 않을 것이다. 내 경우에는 /usr 을 /usr-local 로 mv 명령을 이용해서 변경한 후 다시 /usr 디렉토리를 만들었다. 그리고 나서 다음과 같이 마운트 명령을 내려서 nfs로 마운트가 되는지를 확인하자.

```
mount -t nfs node1:/home /home
```

마운트가 된다면 /home 디렉토리의 내용이 node1의 것으로 바뀌어 있을 것이다. 이제 nfs로 node1의 자원을 이용할 수 있으므로 /etc/fstab에 다음의 줄을 추가 해서 부팅시에 자동적으로 마운트를 하도록 설정하자.

<sup>3</sup>node1에서 프로그램 소스를 수정한 후 node3에서 작업중 다시 소스를 수정해야 하는 상황이 벌어질 경우 node1으로 옮겨와서 수정한후 수정한 소스를 ftp를 통해서 업로드 해야 한다면 정말 불편한 일이 아닐 수 없다. 별도의 로그인/로그아웃 과정 없이 node3에서 바로 소스를 수정하고 이것을 어느 노드에서든 사용할 수 있게 설정하는 것이 보다 합리적일 것이다.

```

[~] 하그터미날
[xenus@node1:~]$ ssh node2
Last login: Sat Mar 23 18:50:11 2002 from node1 on pts/0

MINION node2

Last login: Sat Mar 23 18:50:23 2002 from node1
[xenus@node2:~]$ █

[영어][완성][두벌식]

```

```

# NFS
node1:/home      /home          nfs      defaults,hard,intr    0      0
node1:/usr       /usr           nfs      defaults,hard,intr    0      0

```

fstab파일의 설정에 대한 설명은 생략하겠다. 다만 일반 파티션 마운트와 다른 부분에 대해서만 설명하면, 첫부분의 장치명에 node1:/home이라고 적혀 있는 것은 nfs를 통해서 node1의 /home을 마운트할 장치로 사용하겠다는 의미이며, 옵션의 hard는 만일에 network에 문제가 생겼을 경우 hard옵션은 지속적으로 연결 시도를 한다. intr옵션은 node1에 문제가 발생했을 경우 nfs콜을 인터럽트 가능하게 해 연결을 종료시키는데 유용하다. 이렇게 설정해두면 다음부터는 재부팅시 자동으로 node1의 /home, /usr디렉토리를 마운트 한다. 또한 dselect를 통한 패키지의 설치나 업데이트 역시 node1에서만 해주는 것으로 충분하다.

## 2.5 SSH configuration

예전에 작성된 문서들을 찾아보면 모두 노드간의 이동을 위해 rlogin을 사용한다고 나와있다. 하지만 내가 설치한 Debian(woody) GNU/Linux의 경우에는 rlogin이 ssh로 심볼릭 링크되어 있었다. 그래서 노드간의 접속을 위해 rlogin이 아닌 ssh를 사용했으며 후에 생각해 봐도 r 계열 명령이 아닌 ssh를 쓴것이 보안상으로도 더 훌륭한 선택이었다. 현재는 ssh가 설치되어 있지 않으므로 node1에 ssh를 설치하도록 하자.

ssh를 이용하면 노드간에 인증키를 사전에 교환하여 암호입력 없이 로그인 가능하다. ssh의 인증키 생성에 대해서는 여기서 적는 것 보다 기존에 나와있는 문서가 훌륭하므로 그것을 참조하길 바란다. 문서의 URL은 다음과 같다.

<http://kldp.org/KoreanDoc/html/SSH-KLDP/x80.html>

KLDP의 임은재님이 작성한 문서로 대단히 자세히 설명되어 있다. 다음은 ssh 인증키 교환으로 인해 인증없이 node1에서 node2로의 접속 장면을 캡처한 것이다.

이전 섹션까지 아무런 문제없이 성공했다면 이제 각 노드들을 하나로 묶어 단일 클러스터 시스템으로 동작하게 하기 위한 모든 준비가 끝난 것이다.

## 제 3 절 LAM/MPI configuration

### 3.1 LAM/MPI는 무엇인가?

클러스터는 기본적으로 각 노드마다 별도의 메모리를 갖고 있다. 따라서 우리가 설정한 node1에서 node2나 다른 노드와 데이터를 주고 받으려면 메모리 상에서의 직접 공유나 복사가 불가능 하므로 network을 통해서 이루어 진다. 따라서 클러스터는 이러한 메세지 전달구조가 선행되어야 하며 클러스터를 완벽하게 사용하기 위해서는 이러한 전달구조에 기반하여 프로그램을 작성해야 한다.

다행이도 이러한 메세지 전달을 위한 라이브러리가 이미 작성되어서 배포되고 있다. 대표적으로 PVM이나 MPI가 그것이다. 여기서 각자의 특징에 대해 자세하게 논의하는 것은 불필요한 노력이므로 생략하도록 하겠다. 다만 간략하게 설명하면 PVM이 역사적으로 먼저 나와서 많은 병렬컴퓨터들에서 사용되어 왔으며 MPI는 PVM보다 늦게 나오긴 했지만 더 많은 기능을 지니고 있으며 하드웨어 벤더들의 지원을 받고 있다. MPI는 MPICH와 LAM/MPI라는 두가지 버전이 있는데 우리는 그중 LAM/MPI를 사용할 것이다. 여기서 언급한 각 라이브러리들은 다음의 URL에서 정보를 얻을 수 있다.

PVM :  
<http://www.epm.ornl.gov/pvm/>  
MPICH :  
<http://www-unix.mcs.anl.gov/mpi/mpich/>  
LAM/MPI :  
<http://www.lam-mpi.org/>

데비안 시스템에서 LAM/MPI는 dselect 리스트의, lam3와 lam3-dev 두 패키지를 선택해서 설치하는 것으로 모든 설치가 끝난다.

### 3.2 lamboot으로 사용할 노드 선택하기

lam은 기본적으로 각 node를 인식하고 관리하는 데몬이 먼저 실행되도록 되어 있다. lam데몬은 lamboot 이라는 명령으로 실행할 수 있는데 lamboot은 lamhosts 파일을 참조하여 자신이 사용할 node들을 인식한다. lamhosts파일에는 간략하게 사용할 노드들의 이름을 적어주면 된다. 내가 사용할 시스템은 4개의 노드를 모두 사용할 계획이므로

```
node1  
node2  
node3  
node4
```

이라는 내용을 적어주면 된다. 만일 계산을 하는데 두개 정도의 CPU만 사용하면 되고 자신 이외에 다른 사람이 어떤 노드를 써야 한다면 여기서 사용할 노드를 취사 선택할 수 있다.

실행은 다음과 같다. -v 옵션은 디버그 정보를 보여준다는 뜻으로 사용하고자 하는 node들이 제대로 인식되는지 확인할 수 있다. 다음은 lamboot명령으로 각 노드들을 제대로 인식할경우 나타나는 화면이다.

정상적으로 인식이 된다면 성공한 것이다. 이제 LAM/MPI 라이브러리를 이용해서 프로그램을 작성하기만 하면 훌륭한 계산용 클러스터를 이용할 수 있다.

### 3.3 LAM/MPI programming

현재 이 문서에서 제작한 클러스터 minion 기반으로 FDTD method를 이용한 photonic band gap calculator를 개발중이다. 아직 문서화 할만큼 충분한 프로그래밍 스킬을 얻지 못해 문서화는 늦어지고 있지만 pbc의 첫번째 버전이 나오는데로 LAM/MPI 프로그래밍에 대한 문서를 작성할 예정이다.

```
하그터미날
[xenus@node1:~]$ lamboot -v lamhosts
LAM 6.5.6/MPI 2 C++/ROMIO - University of Notre Dame
Executing hboot on n0 (node1 - 1 CPU)...
Executing hboot on n1 (node2 - 1 CPU)...
Executing hboot on n2 (node3 - 1 CPU)...
Executing hboot on n3 (node4 - 1 CPU)...
topology done
[xenus@node1:~]$ █
```

[영어][완성][두벌식]

## 제 4 절 References

<http://kldp.org/Translations/html/ClusterQuickStart-KLDP/ClusterQuickStart-KLDP.htmltoc14>  
<http://www.lam-mpi.org/>  
<http://www.phyast.pitt.edu/beowulf/>  
<http://kldp.org/HOWTO/html/Beowulf/Beowulf-HOWTO.htmltoc6>  
<http://kldp.org/HOWTO/html/Parallel-Processing/Parallel-Processing-HOWTO.htmltoc3>