SLURM 관리자/이용자 가이드

Ver. 0.4

2014년 12월

슈퍼컴퓨팅서비스센터

KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information
www.kisti.re.kr

목 차

1. 시스템 사양 및 구성

2. 사전 준비 작업

가. 패키지 설치

나. **Munge** 설정

3. SLURM 설치

가. **SLURM** 소스 컴파일 및 설치

나. **SLURM** 환경 설정

다. **SLURM** 실행

4. Accounting 설치

가. 텍스트 파일 기반 설정

나. **SlurmDBD** 설정 및 실행

다. **SlurmDBD** 연동을 위한 **MySQL** 설정

5. 가속 장치 설정

가. **GPU** 인식을 위한 **SLURM** 설정

나. **Intel Xeon Phi Coprocessor** 인식을 위한 **SLURM** 설정

6. 사용자 작업 스크립트 작성

가. **OpenMP** 병렬 프로그램을 위한 작업 스크립트

나. **MPI** 병렬 프로그램을 위한 작업 스크립트

다. **Hybrid(OpenMP+MPI)** 병렬 프로그램을 위한 작업 스크립트

라. **Offloading + OpenMP** 프로그램을 위한 작업 스크립트

마. **Offloading+OpenMP+MPI** 프로그램을 위한 작업 스크립트

바. **Symmetric** 프로그램을 위한 작업 스크립트

1. 시스템 사양 및 구성

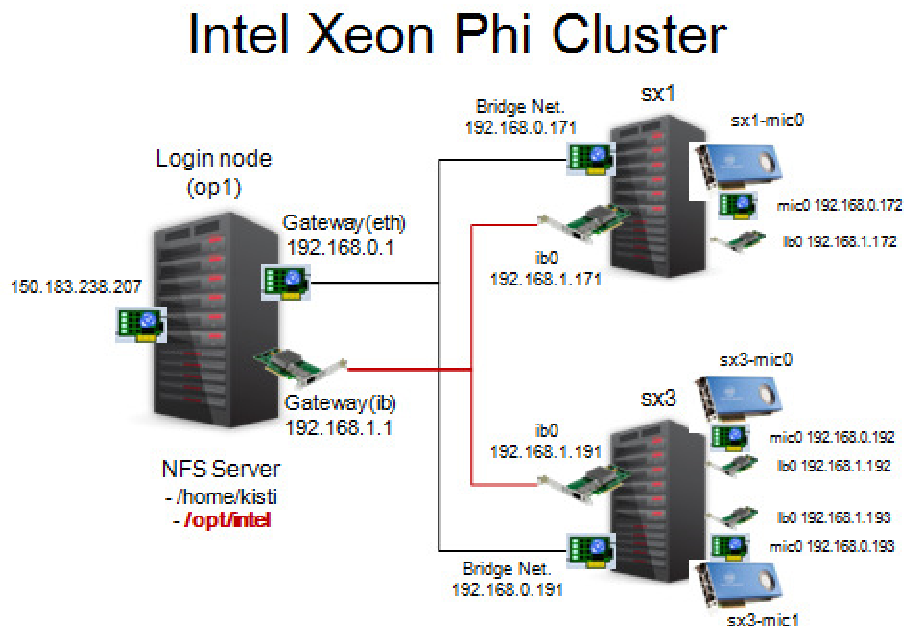
본 문서에서는 아래와 같은 시스템 환경에서 Slurm을 설치하는 방법에 대하여 소개하고 있으나 VirtualBox 또는 VMware 등 가상화 도구를 이용하여 운영체제를 설치한 경우에도 동일하게 적용이 가능하다.

본 문서에서 테스트한 시스템 사양은 다음과 같다.

시스템명	CPU	cores/ node	Memory/ node	OS	비고
op1	AMD Opteron	24	32GB	Oracle Linux 6.5	로그인 노드
intel[1-4]	Intel Xeon	8	32GB	Oracle Linux 6.1	intel 1/2 : HD 6970 intel 3/4 : GTX 580
amd[1-4]	AMD Opteron	24	32GB	Oracle Linux 6.1	amd 1/2 : HD 6970 amd 3/4 : GTX 580
apu[1-8]	AMD LIano	4	16GB	Oracle Linux 6.1	apu 3/5 : GT 520 apu[1-8] : HD 6970

[Slurm 테스트 시스템 사양]

또한, Intel Xeon Phi Coprocessor 장치를 테스트한 환경은 아래의 그림과 같다.



[Slurm 테스트 시스템 구성도]

2. 사전 준비 작업

가. 의존성 패키지 설치

- 리눅스를 설치할 때 최소한의 설치(minimum package option)를 선택하였다고 가정할 때 추가로 설치해야 하는 패키지는 다음과 같다.

1) 공통 추가

```
$ yum install wget gcc openssh-clients
```

2) munge 설치에 필요한 패키지 추가

```
$ yum install rpm-build bzip2-devel
$ yum install openssl-devel zlib-devel
```

3) Slurm 설치에 필요한 패키지 추가

```
$ yum install readline-devel pam-devel
$ yum install perl perl-ExtUtils-MakeMaker
```

나. munge 설치

- munge 설치 파일을 다운로드 받고 rpmbuild로 rpm file을 생성한 후 설치한다.
  (문서 작성 시점에서의 안정 버전은 0.5.11- 1)

1) munge 파일 다운로드

```
$ wget https://munge.googlecode.com/files/munge-0.5.11.tar.bz2
```

2) rpm 파일 생성

```
$ rpmbuild -tb --clean  munge-0.5.11.tar.bz2
```

- 컴파일이 정상적으로 완료되면 /root/rpmbuild/RPMS/x86_64 폴더에 munge 관련 rpm 파일들이 생성되어 있음을 확인할 수 있다.

3) munge rpm 설치

```
$ cd /root/rpmbuild/RPMS/x86_64
$ rpm -ivh munge-0.5.11-1.el6.x86_64.rpm
$ rpm -ivh munge-libs-0.5.11-1.el6.x86_64.rpm
$ rpm -ivh munge-devel-0.5.11-1.el6.x86_64.rpm
```

4) key 생성

```
$ dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
1024+0 records in
1024+0 records out
1024 bytes (1.0kB) copied, 0.00589711s, 174 kB/s
```

- munge key를 생성할 때 다양한 방법이 있으나 /dev/random은 시간이 많이 소요되므로 pseudo random인 /dev/urandom 정도로 생성하면 무난하다.

다. munge 설정

- munge 관련하여서는 로그 또는 실행 프로세스 관련하여 key 값을 안전하게 보호해야 하므로 폴더에 대한 권한 설정이 주를 이룬다. 아래는 관련된 폴더에 대한 권한 설정 정보를 포함하고 있다.

```
$ chmod 700 /etc/munge
$ chmod 400 /etc/munge/munge.key
$ chown -R munge:munge /etc/munge
$ chmod 700 /var/log/munge
$ chown -R munge:munge /var/log/munge
$ chmod 711 /var/lib/munge
$ chown -R munge:munge /var/lib/munge
$ chmod 755 /var/run/munge
$ chown -R munge:munge /var/run/munge
```

3. SLURM 설치

가. SLURM 소스 컴파일 및 설치

- slurm 설치 파일을 다운로드 받고 rpmbuild로 rpm file을 생성한 후 설치한다. (문서 작성 시점에서의 안정 버전은 2.6.6- 2)

```
$ wget http://www.schedmd.com/download/latest/slurm-2.6.6-2.tar.bz2
```

- rpmbuild로 rpm 파일을 생성하면 /root/rpmbuild/RPMS/x86_64 폴더에 slurm 관련 rpm 파일들이 생성된다.

```
$ rpmbuild -ta slurm-2.6.6-2.tar.bz2
$ cd /root/rpmbuild/RPMS/x86_64
$ rpm -ivh slurm-plugins-2.6.6-2.el6.x86_64.rpm
$ rpm -ivh slurm-2.6.6-2.el6.x86_64.rpm
$ rpm -ivh slurm-munge-2.6.6-2.el6.x86_64.rpm
$ rpm -ivh slurm-devel-2.6.6-2.el6.x86_64.rpm
```

- slurm을 실행하기 전 사용자 계정을 생성해야 한다.
  (사용자와 그룹 번호를 500으로 부여하였으나 시스템에 맞도록 수정한다.)

```
$ groupadd --gid 500 slurm
$ echo "slurm:x:500:500:SLURM User:/etc/slurm:/bin/false" >> /etc/passwd
$ chmod 700 /var/log/slurm
$ chown -R slurm:slurm /var/log/slurm
```

나. SLURM 환경 설정

- /etc/slurm 폴더 아래에 예제 파일(slurm.conf.example)이 존재한다. 이 파일을 참고하여 설정을 변경한 후 이용하도록 한다.

```
$ cd /etc/slurm
$ cp slurm.conf.example slurm.conf
```

- slurm.conf 파일을 편집하여 해당 항목을 아래와 같이 설정한다.

```
ClusterName=test_cluster
ControlMachine=op1
ControlAddr=op1
......
SelectType=select/cons_res
SelectTypeParameters=CR_Core
......
SlurmCtldLogFile=/var/log/slurm/slurmctld.log
SlurmdLogFile=/var/log/slurm/slurmd.log
......
# COMPUTE NODES
NodeName=intel[1-4] CPUs=16 RealMemory=32000 Sockets=2
         CoresPerSocket=4 ThreadsPerCore=2 State=UNKNOWN
NodeName=amd[1-4] CPUs=24 RealMemory=32000 Sockets=2
         CoresPerSocket=12 ThreadsPerCore=1 State=UNKNOWN
NodeName=apu[1-6] CPUs=4 RealMemory=16000 Sockets=1
         CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
NodeName=apu[7-8] CPUs=4 RealMemory=15900 Sockets=1
         CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
PartitionName=debug Nodes=intel[1-4],amd[1-4],apu[1-8]
         Default=YES MaxTime=INFINITE State=UP
```

다. SLURM 실행

- munge key 값을 우선 복사하고 데몬을 실행한다.

```
$ pdcp -w intel[1-4],amd[1-4],apu[1-8] /etc/munge/munge.key /etc/munge
$ /etc/init.d/munge start
$ pdsh -w intel[1-4],amd[1-4],apu[1-8] "/etc/init.d/munge start"
```

- munge 데몬이 정상적으로 실행되고 있는지 확인하기 위해서는 /var/log/munge 폴더의 로그파일을 확인하거나 아래의 명령을 실행하여 문제가 발생할 경우 확인할 수 있다.

```
$ munge -n | ssh intel1 unmunge
```

- slurm.conf 파일을 복사하고 slurm 데몬을 실행한다.

```
$ pdcp -w intel[1-4],amd[1-4],apu[1-8] /etc/slurm/slurm.conf /etc/slurm
$ /etc/init.d/slurm start
$ pdsh -w intel[1-4],amd[1-4],apu[1-8] "/etc/init.d/slurm start"
```

- slurm 설정이 정상적으로 되어 있고 데몬이 실행중이라면 아래와 같은 결과를 얻게 된다.

```
$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE   NODELIST
debug*     up     infinite      13  idle    intel[1-4],amd[1-4],apu[1-8]
```

- 만약 state가 down 상태인 노드가 있다면 /var/log/slurm 폴더의 slurmctld.log 로그파일 또는 slurmd.log 파일을 확인한다.
- slurm 데몬을 재시작 할 때는 아래와 같이 캐쉬를 지우는 명령으로 재시작한다.

```
$ /etc/init.d/slurm  stop
$ /etc/init.d/slurm startclean
```

4. Accounting 설치

가. 텍스트 파일 기반 설정

- SLURM 컨트롤을 성공적으로 설정한 경우 텍스트 파일로 로그를 남기고자 하는 경우 slurm.conf 파일만 수정하여도 이용할 수 있다.

```
ClusterName=test_cluster
ControlMachine=op1
ControlAddr=op1
......
JobCompType = jobcomp/filetxt
JobCompLoc=/var/log/slurm/job_completions
# ACCOUNTING
JobAcctGatherType=jobacct_gather/linux
AccountingStorageType=accounting_storage/filetxt
AccountingStorageHost=op1
AccountingStorageLoc=/var/log/slurm/accounting
......
# COMPUTE NODES
NodeName=intel[1-4] CPUs=16 RealMemory=32000 Sockets=2
         CoresPerSocket=4 ThreadsPerCore=2 State=UNKNOWN
NodeName=amd[1-4] CPUs=24 RealMemory=32000 Sockets=2 C
         CoresPerSocket=12 ThreadsPerCore=1 State=UNKNOWN
NodeName=apu[1-6] CPUs=4 RealMemory=16000 Sockets=1
         CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
NodeName=apu[7-8] CPUs=4 RealMemory=15900 Sockets=1
         CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
PartitionName=debug Nodes=intel[1-4],amd[1-4],apu[1-8]
         Default=YES MaxTime=INFINITE State=UP
```

- 굵은색으로 표시한 부분을 추가하고 데몬을 다시 실행한 후 간단한 예제를 실행한 후의 결과는 다음과 같다.

```
$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE   NODELIST
debug*    up    infinite     13  idle    intel[1-4],amd[1-4]
```

나. SlurmDBD 설치 및 실행

- SlurmDBD 데몬을 이용하지 않고 DBMS에 바로 정보를 기록할 수 있으나 이 경우 accounting을 이용하는 모든 명령어가 사용자와 암호 정보를 공유해야 하기 때문에 보안상 취약해진다. 따라서, SLURM에서는 SlurmDBD 데몬을 이용하여 동일한 효과를 얻는 방법을 적용하였다.
- 현재 SLURM에서 지원하는 DB는 MySQL과 MariaDB이다. 본 문서에서는 MySQL과의 연동 방법을 다룬다.
- SlurmDBD를 MySQL과 함께 이용하기 위해서는 아래와 같이 컴파일하고 설치한다.

```
$ yum install mysql-server mysql-devel
$ tar xvfj slurm-2.6.6-2.tar.bz2
$ cd slurm-2.6.6-2
$ vi slurm.spec
%slurm_without_opt mysql => %slurm_with_opt mysql
$ cd ..
$ tar cvfj slurm-2.6.6-2-modified.tar.bz2 slurm-2.6.6-2
$ rpmbuild -ta slurm-2.6.6-2-modified.tar.bz2
$ rpm -ivh /root/rpmbuild/RPMS/x86_64/slurm-sql-2.6.6-2.el6.x86_64.rpm
$ rpm -ivh /root/rpmbuild/RPMS/x86_64/slurm-slurmdbd-2.6.6-2.el6.x86_64.rpm
```

- MySQL을 이용하기 전 my.cnf 파일에 아래와 같이 설정을 추가한다.

```
$ vi /etc/my.cnf
[mysqld]
innodb_buffer_pool_size = 64M   <= 최소 64M 이상
[mysqld_safe]
......
```

- /etc/slurm/slurm.conf에서는 아래와 같이 설정한다. 그리고, example 파일을 복사하여 slurmdbd.conf 파일을 생성하고 설정을 변경한다.

```
$ vi /etc/slurm/slurm.conf
AccountingStorageType=accounting_storage/slurmdbd
$ cp /etc/slurm/slurmdbd.conf.example slurmdbd.conf
$ chmod 600 /etc/slurm/slurmdbd.conf
```

```
$ vi /etc/slurm/slurmdbd.conf
DbdAddr=op1
DbdHost=op1
SlurmUser=slurm
......
DebugLevel=4
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd.pid
......
StorageType=accounting_storage/mysql
StorageHost=op1
StorageUser=slurm         <= database 사용자명
StoragePass=kisti123!     <= database 암호설정
StorageLoc=slurm_acct_db <= database 이름
```

다. SlurmDBD 연동을 위한 MySQL 설정

- MySQL에 slurm 사용자와 권한을 설정하고 데이터베이스를 생성한다.

```
$ mysql
mysql> grant all on slurm_acct_db.* to 'slurm'@'localhost'
       identified by 'kisti123!';
mysql> grant all on slurm_acct_db.* to 'slurm'@'op1'
       identified by 'kisti123!';
mysql> create database slurm_acct_db;
mysql> quit
$
```

- sacctmgr 명령어를 이용하여 SLURM에서 설정한 클러스터와 데이터베이스를 연결한다.

```
$ /etc/init.d/slurmdbd start
$ sacctmgr add cluster test_cluster
 Adding Cluster(s)
  Name                = test_cluster
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
$ sacctmgr add account none,test Cluster=test_cluster
Description= “none”  organization= “KISTI”
  Adding Account(s)
    none
    test
 Settings
    Description        =  none
    Organization      = kisti
  Associations
```

```
    A = none            C = test_clust
    A = test              C = test_clust
  Settings
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
```

- 마지막으로 MySQL에 접속하여 테이블이 생성되었는지 확인하고 간단한 예제를 실행한 후 결과를 확인한다.

```
$ mysql
mysql> use slurm_acct_db;
mysql> show tables;
+---------------------------------------+
| Tables_in_slurm_acct_db               |
+---------------------------------------+
| acct_coord_table                      |
| acct_table                            |
| cluster_table                         |
| test_cluster_assoc_table              |
| test_cluster_assoc_usage_day_table    |
……
| qos_table                             |
| table_defs_table                      |
| txn_table                             |
| user_table                            |
+---------------------------------------+
mysql> quit
$ srun -N2 -l /bin/hostname
$ sacct
 JobID   JobName   Partition  Account   AllocCPUS  State  ExitCode
 -----   -------   ---------  -------   --------   -------  -------
 2      hostname   debug                       2   COMPLETED     0:0
$
```

5. 가속기 장치 설정

가. GPU 인식을 위한 SLURM 설정

- /dev/nvidia[n] n=0,1,2,... 와 같이 NVIDIA GPU 디바이스 설치 및 인식은 다른 문서를 참조하여 설정하도록 한다.
- slurm.conf 파일에 아래와 같이 두 부분을 추가한다.

```
......
# COMPUTE NODES
GresTypes=gpu
NodeName=intel[1-2] CPUs=16 RealMemory=32000 Sockets=2
         CoresPerSocket=4 ThreadsPerCore=2 State=UNKNOWN
NodeName=intel[3-4] CPUs=16 RealMemory=32000 Sockets=2
         CoresPerSocket=4 ThreadsPerCore=2 Gres=gpu:1 State=UNKNOWN
......
PartitionName=debug Nodes=intel[1-4],amd[1-4],apu[1-8]
         Default=YES MaxTime=INFINITE State=UP
```

- slurm manager 즉, slurm.conf에서 ControlMachine으로 설정한 노드의 /etc/slurm 폴더에 아래와 같이 gres.conf 파일을 생성한다.

```
NodeName=intel[3-4]  Name=gpu  File=/dev/nvidia0
```

- GPU 카드가 장착되어 있는 노드에 아래와 같이 gres.conf 파일을 설정한다.

```
Name=gpu  File=/dev/nvidia0
```

- GPU 카드가 장착되어 있지 않은 노드에서 명령을 실행하여도 GPU

자원을 인식하여 노드를 할당한다. 아래는 실행 방법 및 결과 화면이다.

```
 $ srun --gres=gpu:1 -n1
/usr/local/cuda/samples/0_Simple/cdpSimplePrint/cdpSimplePrint
 Starting Simple Print (CUDA Dynamic Parallelism)
 GPU 0 (GeForce GTX 580) does not support CUDA Dynamic Parallelism
 cdpSimplePrint requires GPU devices with compute SM 3.5 or higher.
Exiting...
 $
```

나. Intel Xeon Phi Coprocessor 인식을 위한 SLURM 설정

- 인텔의 Coprocessor인 Xeon Phi가 설치되었다면 /dev/mic[n] n=0,1,2,... 와 같이 디바이스 장치가 보이게 된다. Intel Xeon Phi 카드 인식 및 디바이스 드라이버 설치는 다른 문서를 참조하여 설정하도록 한다.
- slurm.conf 파일에 아래와 같이 두 부분을 추가한다.
  (sx1, sx3 노드는 Xeon Phi 카드가 각각 1장, 2장씩 장착되어 있다.)

```
......
# COMPUTE NODES
GresTypes=mic
NodeName=sx1 CPUs=32 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2
         State=UNKNOWN  Gres=mic:1
NodeName=sx3 CPUs=32 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2
         State=UNKNOWN  Gres=mic:2
......
PartitionName=debug Nodes=sx[1, 3] Default=YES MaxTime=INFINITE
               State=UP
```

- slurm manager 즉, slurm.conf에서 ControlMachine으로 설정한 노드의 /etc/slurm 폴더에 아래와 같이 gres.conf 파일을 생성한다.

```
NodeName=sx1  Name=mic  File=/dev/mic0
NodeName=sx3  Name=mic  File=/dev/mic0
NodeName=sx3  Name=mic  File=/dev/mic1
```

- Intel Xeon Phi 카드가 장착되어 있는 노드에 아래와 같이 gres.conf 파일을 설정한다.
  (아래의 예는 sx3 노드로 Xeon Phi 카드 2개가 장착되어 있어서 mic0, mic1 두 개를 적어준 것이다.)

```
Name=mic  File=/dev/mic0
Name=mic  File=/dev/mic1
```

- Intel Xeon Phi 카드가 장착되어 있지 않은 노드에서 명령을 실행하여도 자원을 인식하여 노드를 할당한다. 아래는 실행 방법 및 결과 화면이다.

```
$ srun –gres=mic:1 -n1 /bin/hostname
sx1
$ srun –gres=mic:2 -n1 /bin/hostname
sx3
$ srun –gres=mic:2 -n1 micctrl -s
mic0: online (mode: linux image: /usr/share/mpss/boot/bzImage-knightscorner)
mic1: online (mode: linux image: /usr/share/mpss/boot/bzImage-knightscorner)
$
```

6. 사용자 작업 스크립트 작성

- 병렬 프로그램 중 OpenMP, MPI, Hybrid(OpenMP+MPI) 방식에 대한 간단한 예제와 작업 스크립트 파일 작성을 소개한다.
- 모든 코드는 C 언어로 작성하였으며 Intel Xeon Phi 카드에서 실행할 바이너리를 생성하는 예제도 포함하여 컴파일은 Intel 컴파일러를 사용하여 컴파일한다.

가. OpenMP 병렬 프로그램을 위한 작업 스크립트

1) OpenMP 예제 프로그램

```
#include <stdio.h>
#include <omp.h>
void main()
{
#pragma omp parallel
{
    printf("tid=%d₩n", omp_get_thread_num());
}
}
```

2) OpenMP 프로그램 작업 스크립트 예제

```
#!/bin/sh
#SBATCH -J omp
#SBATCH -o omp_%j.out
#SBATCH -e omp_%j.err
#SBATCH --cpus-per-task=4
export OMP_NUM_THREADS=4
./bin/omp.x
```

- SLURM의 작업 스크립트에는 많은 옵션으로 설정할 수 있으나 대표적으로 세가지만 언급한다.
- #SBATCH 키워드는 SLURM 스케줄러가 인식하는 옵션을 설정할 때 이용한다. -J 옵션은 작업의 이름을 지정할 때 사용하고, -o 옵션은 출력 파일명을 명시할 때 사용한다. -e 옵션은 에러를 저장할 파일을 지정하는 옵션으로 예제에서는 omp_%j.out/err를 지정하였다. %j는 SLURM 스케줄러에서 지정하는 작업의 ID 값을 반영하는 것으로 31번 작업인 경우 omp_31.out/err로 설정된다.
- #SBATCH —cpus-per-task 옵션은 CPUs/Task로 각각의 태스크(프로세스)에 대해 몇 개의 CPU 자원을 할당한 것인지를 설정하는 것이다. export OMP_NUM_THREADS도 프로세스에 CPU를 할당하는 옵션인데 전자의 경우 스케줄러에게 정보를 알려주는 것이고 후자의 경우 프로세스에 전달되는 환경 설정값이다.
- 따라서, #SBATCH —cpus-per-task=4를 설정하지 않아도 프로그램이 4개의 쓰레드를 생성하여 결과를 얻게 되지만 export OMP_NUM_THREADS=4를 생략하면 할당된 노드에 있는 코어의 개수만큼 쓰레드가 할당되거나 기존에 다른 설정값이 적용되어 있다면 해당 정보가 반영되어 쓰레드가 생성된다.

3) OpenMP 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ icc -openmp -o bin/omp.x src/omp.c
[dokto76@sx3 test]$ sbatch omp.cmd
Submitted batch job 1694
[dokto76@sx3 test]$ cat omp_1694.out
tid=0
tid=1
tid=2
tid=3
```

- sbatch 명령을 이용하여 작업 스크립트를 배치 형태로 실행하면 위의 예제의 경우 작업 ID가 1694로 할당되어 결과 정보는 omp_1694.out 파일에 저장된다.

```
[dokto76@sx3 test]$ sacct
JobID    JobName  Partition    Account  AllocCPUS    State ExitCode
------  -------  ---------  --------  ---------    -------------
1694      omp       basic                       4   COMPLETED      0:0
1694.batch          batch                       1   COMPLETED      0:0
[dokto76@sx3 test]$
```

- 프로그램이 정상적으로 종료되었음을 확인하는 방법은 4장의 Account 설정에서 파일 기반 또는 데이터베이스를 이용하여 로그를 저장하면 sacct라는 명령어로 확인할 수 있다.

나. MPI 병렬 프로그램을 위한 작업 스크립트

1) MPI 프로그램 예제

```
#include <stdio.h>
#include <mpi.h>
void main(int argc, char *argv[])
{
    int rank, nprocs, namelen;
    char name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &namelen);
    printf("rank %d of %d on %s₩n", rank, nprocs, name);
    MPI_Finalize();
}
```

2) MPI 프로그램 작업 스크립트 예제

```
#!/bin/sh
#SBATCH -J mpi
#SBATCH -o mpi_%j.out
#SBATCH -e mpi_%j.err
## 4 processes & 2 processes/node
#SBATCH -n 4
#SBATCH --tasks-per-node=2

mpirun ./bin/mpi.x
```

- MPI 프로그램을 실행할 때 필요한 옵션은 프로세스 수이다.
- #SBATCH –n 4는 4개의 프로세스를 생성해야 함을 의미하고 #SBATCH —tasks-per-node=2는 하나의 노드에 몇 개의 프로세스를 할당할 것인지 지정하는 옵션으로 4개의 프로세스를 생성하는데 노드당 2개를 할당하므로 2개의 노드를 이용하게 된다.

3) MPI 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ mpiicc  -o  bin/mpi.x  src/mpi.c
[dokto76@sx3 test]$ sbatch mpi.cmd
Submitted batch job 1695
[dokto76@sx3 test]$ cat mpi_1695.out
rank 2 of 4 on sx3
rank 3 of 4 on sx3
rank 0 of 4 on sx1
rank 1 of 4 on sx1
[dokto76@sx3 test]$
```

- 4개의 프로세스를 노드당 2개씩 할당하도록 설정하여 sx1, sx3 노드에 각각 2개의 프로세스가 실행되었음을 알 수 있다.

다. Hybrid(OpenMP+MPI) 병렬 프로그램을 위한 작업 스크립트

1) Hybrid 프로그램 예제

```
#include <stdio.h>
#include <omp.h>
#include <mpi.h>
void main(int argc, char *argv[])
{
    int rank, nprocs, namelen, provided;
    char name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &namelen);
#pragma omp parallel
{
    printf("tid %d rank %d of %d on %s₩n",
                omp_get_thread_num(), rank, nprocs, name);
}
    MPI_Finalize();
}
```

- Hybrid 프로그램은 MPI 병렬 라이브러리를 이용하여 통신을 하면서 하나의 노드에서는 SMP(Symmetric Multi-processing) 방식으로 자원을 이용할 때 지정하는 방식이다.
- MPI_Init_thread 함수에서 MPI_THREAD_MULTIPLE 옵션으로 설정을 하면 각각의 쓰레드 내에서 MPI 통신이 가능하게 된다. 보다 자세한 사항은 KISTI 슈퍼컴퓨팅교육센터 홈페이지에 있는 자료를 참고하도록 한다.

2) Hybrid 프로그램 작업 스크립트 예제

```
#!/bin/sh
#SBATCH -J omp_mpi
#SBATCH -o omp_mpi_%j.out
#SBATCH -e omp_mpi_%j.err
# 2 processes
# 1 process/node, 3 threads/process
#SBATCH -n 2
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=3
export OMP_NUM_THREADS=3

mpirun ./bin/hybrid.x
```

- Hybrid 프로그래밍은 OpenMP와 MPI 프로그램을 동시에 이용하는 방식으로 작업 스크립트도 각각의 프로그램 방식에서 이용한 옵션을 모두 포함하여 작성을 한다.
- #SBATCH –n 2는 2개의 프로세스를 생성해야 함을 의미하고 #SBATCH —tasks-per-node=1는 각각의 노드에 하나의 프로세스만이 지정되도록 하여 위의 예제는 총 2개의 노드를 이용하게 된다. 각각의 프로세스는 #SBATCH —cpus-per-task와 export OMP_NUM_THREADS=3에 의해 3개의 쓰레드를 생성하게 된다.

3) Hybrid 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ mpiicc  -openmp  -o  bin/hybrid.x  src/hybrid.c
[dokto76@sx3 test]$ sbatch hybrid.cmd
Submitted batch job 1697
[dokto76@sx3 test]$ cat hybrid_1697.out
tid 0 rank 1 of 2 on sx3
```

```
tid 1 rank 1 of 2 on sx3
tid 2 rank 1 of 2 on sx3
tid 0 rank 0 of 2 on sx1
tid 2 rank 0 of 2 on sx1
tid 1 rank 0 of 2 on sx1
[dokto76@sx3 test]$
```

- 따라서 위의 작업 스크립트를 실행하게 되면 2개의 노드가 할당되어 노드당 하나의 프로세스가 실행하게 되고 각각의 프로세스마다 세 개의 쓰레드를 실행하게 되어 총 6개의 쓰레드가 생성되어 정보를 출력하고 종료하게 된다.

라. 오프로딩(Offloading) + OpenMP 프로그램을 위한 작업 스크립트

1) 오프로딩 + OpenMP 프로그램 예제

```
#include <stdio.h>
#include <omp.h>
void main()
{
#pragma omp parallel
{
    printf("host tid=%d\n", omp_get_thread_num());
}
#pragma offload target(mic)
#pragma omp parallel
{
    printf("xeon phi tid=%d\n", omp_get_thread_num());
}
}
```

- Intel Xeon Phi 카드를 사용하는 방법으로는 오프로딩 방식과 Native 방식이 있다. 이 중 오프로딩 방식은 디렉티브(Directive)를 이용하여 쉽게 코딩할 수 있는데, 인텔에서 제공하는 #pragma offload 키워드를 통해 Xeon Phi 카드에서 실행할 부분을 지정할 수 있다.[1] 보다 자세한 사항은 Intel 홈페이지에 있는 자료를 참고하도록 한다.

2) 오프로딩 + OpenMP 프로그램 작업 스크립트 예제

```
#!/bin/bash
#SBATCH -J offload
#SBATCH -o offload_%j.out
#SBATCH -e offload_%j.err
#SBATCH --gres=mic
export OMP_NUM_THREADS=4
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=2
./bin/offload.x
```

- #SBATCH —gres=mic 옵션을 통해 Xeon Phi 카드가 장착되어 있는 노드를 스케줄러가 선택하게 된다. 테스트 시스템에서는 sx1 노드가 선택이 되는데 Xeon Phi 카드가 2개 있는 노드를 이용하기 위해서는 #SBATCH —gres=mic:2로 카드수를 지정하면 sx3 노드가 선택된다.
- MIC_ENV_PREFIX, MIC_OMP_NUM_THREADS는 Xeon Phi 카드에 필요한 옵션 설정으로 MIC_ENV_PREFIX는 환경설정 변수 중에서 MIC_로 시작하는 환경변수는 Xeon Phi 카드에 적용됨을 의미하고 MIC_OMP_NUM_THREADS는 MIC_를 제외하면 쓰레드 수를 명시하는 환경변수로 Xeon Phi 카드에서 실행할 쓰레드 수를 나타낸다.

1) OpenMP 4.0부터 GPU, Xeon Phi 등 가속기 장치를 이용할 수 있는 디렉티브도 포함하고 있음

3) 오프로딩 + OpenMP 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ icc  -openmp  -o  bin/offload.x  src/offload.c
[dokto76@sx3 test]$ sbatch offload.cmd
Submitted batch job 1699
[dokto76@sx3 test]$ cat offload_1699.out
xeon phi tid=0
xeon phi tid=1
host tid=0
host tid=1
host tid=2
host tid=3
[dokto76@sx3 test]$
```

- 위의 작업 스크립트를 실행하게 되면 Xeon Phi 카드가 장착되어 있는 노드가 할당되어 CPU에서는 4개의 쓰레드가 실행하게 되고 Xeon Phi 카드에서는 두 개의 쓰레드를 실행하게 되어 총 6개의 쓰레드가 생성되어 정보를 출력하고 종료하게 된다.

마. 오프로딩(Offloading) + OpenMP + MPI 프로그램을 위한 작업 스크립트

1) 오프로딩 + OpenMP + MPI 프로그램 예제

```
#include <stdio.h>
#include <omp.h>
#include <mpi.h>
void main(int argc, char *argv[])
{
    int rank, nprocs, namelen, provided;
    char name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &namelen);
#pragma omp parallel
{
    printf("tid %d rank %d of %d on %s₩n",
                omp_get_thread_num(), rank, nprocs, name);
}
#pragma offload target(mic)
#pragma omp parallel
{
    printf("xeon phi tid %d rank %d of %d on %s₩n",
                omp_get_thread_num(), rank, nprocs, name);
}
    MPI_Finalize();
}
```

- 호스트에서 생성한 프로세스들 간의 통신에 MPI 라이브러리를 이용하고 호스트 프로세스와 Intel Xeon Phi 카드는 오프로딩

방식으로 가속기 자원을 이용하는 방법으로 오프로딩+OpenMP 방식에서 MPI 라이브러리 사용이 추가된 것이다.

2) 오프로딩 + OpenMP + MPI 프로그램 작업 스크립트 예제

```
#!/bin/sh
#SBATCH -J offload_omp_mpi
#SBATCH -o offload_omp_mpi_%j.out
#SBATCH -e offload_omp_mpi_%j.err
# 2 processes, 1 process/node
# 3 host threads/proc., 2 mic threads/proc.
#SBATCH -n 2
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=3
#SBATCH --gres=mic

export MIC_ENV_PREFIX=MIC
export OMP_NUM_THREADS=3
export MIC_OMP_NUM_THREADS=2

mpirun ./bin/offload_omp_mpi.x
```

- 오프로딩, OpenMP, MPI 실행을 위해 필요한 옵션을 모두 이용하는 방식으로 #SBATCH –n 2, #SBATCH —tasks-per-node=1을 통해 MPI 프로세스 2개가 노드당 하나씩 할당되어 총 2개의 노드가 이용되어야 함을 나타내고, #SBATCH —cpus-per-task=3, export OMP_NUM_THREADS=3을 통해 호스트의 프로세스는 3개의 쓰레드를 생성하게 된다. #SBATCH —gres=mic, #SBATCH MIC_ENV_PREFIX=MIC, #SBATCH MIC_OMP_NUM_THREADS=2를 통해 오프로딩 시 Xeon Phi 카드에서는 2개의 쓰레드가 생성된다.

3) 오프로딩 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ mpiicc  -openmp  -o  bin/offload_omp_mpi.x ₩
>  src/offload_omp_mpi.c
[dokto76@sx3 test]$ sbatch offload_omp_mpi.cmd
Submitted batch job 1703
[dokto76@sx3 test]$ cat offload_omp_mpi_1703.out
tid 0 rank 1 of 2 on sx3
tid 1 rank 1 of 2 on sx3
tid 2 rank 1 of 2 on sx3
tid 0 rank 0 of 2 on sx1
tid 1 rank 0 of 2 on sx1
tid 2 rank 0 of 2 on sx1
xeon phi tid 0 rank 1 of 2 on sx3
xeon phi tid 1 rank 1 of 2 on sx3
xeon phi tid 0 rank 0 of 2 on sx1
xeon phi tid 1 rank 0 of 2 on sx1
[dokto76@sx3 test]$
```

- sx1, sx3 두 개의 노드가 할당되어 sx1 노드에서는 rank 0 프로세스가 실행되었고 sx3 노드에서는 rank 1 프로세스가 실행되었음을 알 수 있다. 각각의 프로세스에서 호스트에는 3개의 쓰레드(tid=0~2)가 실행되었고 오프로딩을 통해 Xeon Phi 카드에서는 2개의 쓰레드(xeon phi tid=0~1)가 실행되었음을 알 수 있다.

바. Symmetric 프로그램을 위한 작업 스크립트

1) Symmetric 프로그램 예제

```
#include <stdio.h>
#include <omp.h>
#include <mpi.h>
void main(int argc, char *argv[])
{
    int rank, nprocs, namelen, provided;
    char name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init_thread(&argc,&argv,MPI_THREAD_MULTIPLE, &provided);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(name, &namelen);
#pragma omp parallel
{
    printf("tid %d rank %d of %d on %s₩n",
               omp_get_thread_num(), rank, nprocs, name);
}
    MPI_Finalize();
}
```

- Symmetric 프로그래밍 모델은 오프로딩 방식과 달리 Xeon Phi 카드에서 처음부터 프로세스가 생성되어 호스트 프로세스와 다른 노드의 프로세스 간에 MPI 통신으로 작업을 수행하는 방법이다.
- 따라서, 오프로딩을 제외한 OpenMP+MPI 방식의 코드를 작성하는 것으로 프로그램 실행이 가능하며 컴파일을 통한 Xeon Phi 카드용 바이너리를 생성하게 되고 환경변수로 Xeon Phi 카드에서 실행할 때 필요로 하는 옵션을 설정하게 된다.

2) 오프로딩 + OpenMP + MPI 프로그램 작업 스크립트 예제

```
#!/bin/sh
#SBATCH -J symmetric
#SBATCH -o symmetric_%j.out
#SBATCH -e symmetric_%j.err
#SBATCH -gres=mic
# total 6 processes
# 2 processes on 2 host nodes
# 1 process/node, 3 threads/process
#SBATCH -n 2
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=3
export OMP_NUM_THREADS=3

# 2 xeon phi processes/host process
# 2 threads/xeon phi process
export MIC_PPN=2
export MIC_OMP_NUM_THREADS=2
export I_MPI_MIC=1
export I_MPI_FABRICS=tcp

mpirun-mic -m ./bin/sym.x.mic -c ./bin/sym.x
```

- 호스트에서는 OpenMP, MPI를 이용할 때 필요로 하는 옵션값을 설정하여 두 개의 프로세스를 생성하고 노드당 하나의 프로세스를 할당하여 두 개의 노드를 이용한다. 각각의 프로세스는 세 개의 쓰레드를 생성한다.
- Xeon Phi 카드에서 생성되는 프로세스는 #SBATCH MIC_PPN=2 값을 통하여 노드당 2개의 프로세스를 생성한다. PPN(Processes Per Node)의 의미로 Xeon Phi 카드가 설정되어 있는 노드에서 하나의 카드당 2개의 프로세스를 생성한다. export

MIC_OMP_NUM_THREADS=2는 export MIC_ENV_PREFIX 환경변수를 설정하지 않아도 Xeon Phi 카드에서 생성할 쓰레드 수를 지정하게 된다.

- export I_MPI_MIC=1은 Symmetric 방식에서 사용할 MPI 라이브러리를 Xeon Phi 카드에서 이용할 수 있도록 지정하며 export I_MPI_FABRICS=tcp 옵션은 인피니밴드를 기본적으로 이용하는 Intel MPI 라이브러리에서 인피니밴드가 구성되어 있지 않은 시스템의 경우 TCP를 이용하여 MPI 통신이 이루어지도록 설정하는 것이다.
- mpirun-mic 프로그램은 인텔에서 제공하는 스크립트로 내부를 보면 호스트 이름에 "-mic0"를 붙여서 Xeon Phi 카드에 대한 hostname을 설정한다. -m 옵션은 Xeon Phi 카드에서 실행할 바이너리 정보를 받는 것으로 ./bin/sym.x.mic 프로그램이 Xeon Phi 카드에서 실행하는 프로그램임을 나타낸다.

3) 오프로딩 작업 스크립트 실행 결과

```
[dokto76@sx3 test]$ mpiicc -openmp -o bin/sym.x src/symmetric.c
[dokto76@sx3 test]$ mpiicc -mmic -openmp -o bin/sym.x.mic ₩
> src/symmetric.c
[dokto76@sx3 test]$ sbatch symmetric.cmd
Submitted batch job 1708
[dokto76@sx3 test]$ cat symmetric_1708.out
tid 0 rank 3 of 6 on sx3
tid 1 rank 3 of 6 on sx3
tid 2 rank 3 of 6 on sx3
tid 0 rank 0 of 6 on sx1
tid 2 rank 0 of 6 on sx1
tid 1 rank 0 of 6 on sx1
tid 0 rank 4 of 6 on sx3-mic0
tid 1 rank 4 of 6 on sx3-mic0
tid 0 rank 5 of 6 on sx3-mic0
```

```
tid 1 rank 5 of 6 on sx3-mic0
tid 0 rank 2 of 6 on sx1-mic0
tid 1 rank 2 of 6 on sx1-mic0
tid 0 rank 1 of 6 on sx1-mic0
tid 1 rank 1 of 6 on sx1-mic0
[dokto76@sx3 test]$
```

- Symmetric 프로그램의 실행 결과를 보면 sx1 노드에 rank 0 프로세스가 할당되었고 sx1 노드의 Xeon Phi 카드에 rank 1,2 프로세스가 할당되어 있다.
- 마찬가지로 sx3 노드에 rank 3 프로세스가 할당되었고 sx3 노드의 Xeon Phi 카드에 rank 4, 5번 프로세스가 할당되어 실행되었음을 알 수 있다.
- 프로세스 내에서 생성된 쓰레드는 호스트의 경우 tid=0~2로 세 개씩 생성되었고 Xeon Phi 카드의 경우 tid=0~1로 쓰레드가 두 개씩 생성되었음을 알 수 있다.
- 인텔에서 제공하는 mpirun-mic 스크립트는 하나의 노드에 Xeon Phi 카드가 두 개 이상 장착되어 있는 경우 모든 카드를 이용하여 Symmetric 방식으로 실행하는 방법은 아직 구현되어 있지 않다.

※ 저자 정보

본 지침서는 Slurm 관리자 및 이용자를 위해 제작되었습니다.

이름	소속/연락처
이승민	슈퍼컴퓨팅서비스통합실/**smlee76@kisti.re.kr**
박주원	슈퍼컴퓨팅서비스통합실/**juwon.park@kisti.re.kr**
함재균	슈퍼컴퓨팅서비스통합실/**jaehahm@kisti.re.kr**

version 정보

Ver. 0.1 : 초안 작성 (2014-03-12)

Ver. 0.2 : SlurmDBD 추가 (2014-05-20)

Ver. 0.3 : NVIDIA GPU 설정 추가 (2014-06-03)

Ver. 0.4 : Intel Xeon Phi 설정 추가 (2014-08-08)

Ver. 0.5 : 사용자 작업 스크립트 추가 (2014-08-20)