KISTI NURION SYSTEM
Quick User Guide

(Eng. version)

October 2019

Supercomputing Center

Korea Institute of Science and Technology Information

www.kisti.re.kr

1. NURION System Overview

Section		KNL compute nodes	CPU-only nodes
Model		Cray CS500	
No. of nodes		8,305	132
Peak performance (Rpeak)		25.3 PFlops	0.4 PFlops
Processor	Model	Intel Xeon Phi 7250 (KNL)	Intel Xeon 6148 (Skylake)
	Rpeak per CPU	3.0464 TFLOPS	1.536 TFLOPS
	No. cores per CPU	68	20
	No. CPU per node	1	2
	On-package Memory	16GB, 490GB/s	-
Main memory	Model	16GB DDR4-2400	16GB DDR4-2666
	Configuration	16GB x6, 6Ch per CPU	16GB x12, 6Ch per CPU
	Memory per node(GB)	96GB	192GB
	bandwidth	115.2GB/s	128GB/s
	Total size	778.6TB	24.8TB
High-performance Interconnect	Topology	Fat Tree	
	Blocking Ratio	50% Blocking	
	Switches	278x 48-port OPA edge switches 8x 768-port OPA core switches	
	bandwidth per port	100Gbps	
High-performance file system (Burst Buffer)	No. of servers	DDN IME240 Servers 48ea	
	Disk per server	16x 1.2TB NVMe SSD, 2x 0.45TB NVMe SSD	
	Total size	0.92PB usable	
	bandwidth	20GB/sec per server, 0.8TB/s total	
Parallel file system	File system	Lustre 2.7.21.3	
	Total size	/scratch: 21PB, /home: 0.76PB, /apps: 0.5PB	
	bandwidth	0.3TB/s	
	RAID	RAID6(8D+2P)	

2. User Environment

a. Access

Node Type	Hostname	Available access methods	Time limit for interactive session
login	nurion.ksc.re.kr	ssh, scp, sftp	20 min
data mover	nurion-dm.ksc.re.kr	ssh, scp, sftp, ftp	-

※ To access NURION system, users have to open a ssh connection to the login nodes, for example, “ssh –l [user_id] nurion.ksc.re.kr”.

※ To change your shell, use the “chsh” command.

※ To change your password, use the “passwd” command.

b. Storage

Section	Path	Quota per account	Limit to No. of Files	Policy of Data Purge
Home directory	/home01	64GB	200K	N/A
Scratch directory	/scratch	100TB	1M	Any file not accessed for the last 15 days

- “lfs quota” command will generate a report showing your quota and usage information:

```
$ lfs quota -h /home01
$ lfs quota -h /scratch
```

c. Environment modules

Category	Modules	
Architecture classification modules	• craype-mic-knl • craype-network-opa	• craype-x86-skylake • htop/2.2.0
Cray modules	· cdt/17.10 · cray-ccdb/3.0.3 · cray-cti/1.0.6 · cray-fftw/3.3.6.2 · cray-fftw_impi/3.3.6.2 · cray-impi/1.1.4 · cray-lgdb/3.0.7 · cray-libsci/17.09.1 · craype/2.5.13	· craypkg-gen/1.3.5 · mvapich2_cce/2.2rc1.0.3_noslurm · mvapich2_gnu/2.2rc1.0.3_noslurm · papi/5.5.1.3 · perftools/6.5.2 · perftools-bas/6.5.2 · perftools-lite/6.5.2 · PrgEnv-cray/1.0.2
Compilers	• intel/17.0.5 • intel/18.0.1 • intel/18.0.3 • intel/19.0.1 • intel/19.0.4	• gcc/6.1.0 • gcc/7.2.0 • gcc/8.3.0 • cce/8.6.3 • pgi/18.10 • pgi/19.1
Libraries depending on compilers	· hdf4/4.2.13 · hdf5/1.10.2 · hdf5-parallel/1.10.2 · lapack/3.7.0 · CDO/1.8.2	· ncl/6.5.0 · ncview/2.1.7 · netcdf/4.6.1 · parallel-netcdf/1.10.0 · NCO/4.7.4 · pio/2.3.1
MPI libraries	• fftw_mpi/2.1.5 • fftw_mpi/3.3.7 • impi/17.0.5 • impi/18.0.1 • impi/18.0.3 • impi/19.0.1	• impi/19.0.4 • mvapich2/2.3 • mvapich2/2.3.1 • openmpi/3.1.0 • ime/mvapich-verbs/2.2.ddn1.4
Libraries depending on MPI	· impi/17.0.5 · impi/18.0.1 · impi/18.0.3 · impi/19.0.1	· impi/19.0.4 · mvapich2/2.3 · mvapich2/2.3.1 · openmpi/3.1.0
Intel package	• advisor/17.0.5 • advisor/18.0.1 • advisor/18.0.3	• vtune/17.0.5 • vtune/18.0.1 • vtune/18.0.3

Application SWs	• cmake/3.12.3 • forge/18.1.2 • grads/2.2.0 • gromacs/2016.4 • gromacs/2018.6 • gromacs/5.0.6 • lammps/8Mar18 • lammps/12Dec18 • namd/2.12 • namd/2.13 • PETSc/3.8.4 • ferret/7.4.3	• python/2.7.15 • python/3.7 • IGPROF/5.9.16 • ImageMagick/7.0.8-20 • qe/6.1 • qt/4.8.7 • qt/5.9.6 • R/3.5.0 • siesta/4.1-b3 • siesta/4.0.2 • cp2k/5.1.0 • tensorflow/1.12.0
Virtualization modules	• singularity/2.5.2	• singularity/3.0.1
Commercial SWs	· cfx/v145 · cfx/v170 · cfx/v181 · cfx/v191 · fluent/v145 · fluent/v170 · fluent/v181	· fluent/v191 · gaussian/g16.a03 · gaussian/g16.a03.linda · gaussian/g16.b01.linda · gaussian/g16.c01.linda · lsdyna/mpp · lshyna/smp

d. How to use environmental modules

○ **Mandatory module**

※ User **must** load one of modules (craype-mic-knl or craype-x86-skylake) according to a system to use. If user intends to use knl nodes, type the command below.

```
$ module load craype-mic-knl
```

Otherwise (if user intends to use skl nodes), type the command below.

```
$ module load craype-x86-skylake
```

○ List all available modules

```
$ module avail|av
```

○ Load module(s)

```
$ module add|load [module name]
```

○ Remove/unload module(s)

```
$ module rm|unload [module name]
```

○ List the modules user have loaded

```
$ module list|li
```

○ Rmove/unload all of the loaded modules

```
$ module purge
```

※ Please be careful to use "module purge", because it removes mandatory module (craype-mic-knl or craype-x86-skylake) including the module "craype-network-opa" necessary to use NURION in any case.

※ User can personalize NURION environment by editing .bashrc (adding lines to load modules frequently used) at /home01/[user_id]. Then, user do not need to load those modules each time user log in to NURION.

e. Submitting jobs with PBS

All jobs can only be submitted through the login node. Also, jobs can only be submitted in /scratch directory.

1). Job-submission queues

Node	Qname	No. of nodes	No. of total CPU cores	Wall Clock Limit (hour)	Job submission limit		Resource occupation Limit	
					Job submittion limit per user	Running job limit per user	No. of nodes per job	
							max	min
KNL	exclusive	2600	176,800	unlimited	100	100	2600	1
	normal	4970	337,960	48	40	20	4970	
	long	300	20,400	120	20	10	300	
	flat	180	12,240	48	20	10	180	
	debug	20	1,360		2	2	2	
SKL	commercial	118	4,720	48	6	2	118	
	norm_skl	118	4,720		10	5		

- ○ Most of the queues followexclusive node based charge policy which means all CPU cores on nodes allocated for the job are charged. On the other hand, jobs in “commercial” queue (intended to run commercial softwares such as Gaussian, CFX, Fluent, Abaqus, LS-Dyna, and MSC Nastran) are charged only for the cores that are actually used.
- ○ Due to the hyperthread setting of NURION, users can utilize up to 68 threads per node when using KNL, and up to 40 threads per node with SKL.

2). Submitting and monitoring batch jobs

○ Submitting batch jobs

```
$ qsub <job script>

ex)qsub hello_world.sh
```

In order to perform batch job operation with PBS, user needs to create a job script file using the PBS keywords described below. For more information, please use “man qsub” command.

Variable	Description
PBS_JOBID	The job identifier assigned to the job by the batch system.
PBS_JOBNAME	The job name supplied by the user.
PBS_NODEFILE	The list of nodes assigned to the job
PBS_O_PATH	Value of PATH from submission environment
PBS_O_WORKDIR	The absolute path of the current working directory of the qsub command.
TMPDIR	The job-specific temporary directory for this job

If you create a script by adding "#PBS -W sandbox=PRIVATE", you can check STDOUT and STDERR generated by PBS while your job is running.

※ For the purpose of collecting data for improving user support, it is obligatory to use program information through PBS option as follows. In other words, you must fill out the PBS -A option according to the table below, and submit your work accordingly. (Vaild since April 2019)

[PBS option name table for applications]

Application	PBS option name	Application	PBS option name
ANSYS (CFS, Fluent)	**ansys**	VASP	**vasp**
Abaqus	**abaqus**	Gromacs	**gromacs**
LS-DYNA	**lsdyna**	Charmm	**charmm**
Nastran	**nastran**	Amber	**amber**
Gaussian	**gaussian**	LAMMPS	**lammps**
OpenFoam	**openfoam**	NAMD	**namd**
WRF	**wrf**	Quantum Espresso	**qe**
CESM	**cesm**	QMCpack	**qmc**
MPAS	**mpas**	BWA	**bwa**
ROMs	**roms**	SIESTA	**siesta**
MOM	**mom**	in-house code	**inhouse**
TensorFlow	**tf**	Caffe	**caffe**
PyTorch	**pytorch**	Qchem	**qchem**
grims	**grims**	The others	**etc**

○ Examples of batch scripts

■ Batch script to run a serial job (serial.sh)

```
#!/bin/sh
#PBS -N serial_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

./a.out
```

■ Batch script to run an OpenMP job (omp.sh)

```
#!/bin/sh
#PBS -N openmp_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=1:ncpus=40:ompthreads=40:ompthreads=1
#PBS -l walltime=04:00:00


cd $PBS_O_WORKDIR


./a.out
```

※ Example script occupies one node (select=1) with 40 threads (ompthreads=40)

■ Batch script to run an MPI job using either Intel MPI or OpenMPI (mpi.sh)

```
#!/bin/sh
#PBS -N mvapich2_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=4:ncpus=32:mpiprocs=32:ompthreads=1
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

mpirun ./a.out
```

※ Example scripts occupies four nodes (select=4) with 32 processors per node (total 128 processors)

■ Batch script to run an MPI job using Mvapich2 (mpi_mvapich2.sh)

```
#!/bin/sh
#PBS -N mvapich2_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=4:ncpus=32:mpiprocs=32:ompthreads=1
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

TOTAL_CPUS=$(wc -l $PBS_NODEFILE | awk '{print $1}')

mpirun_rsh -np ${TOTAL_CPUS} -hostfile $PBS_NODEFILE ./a.out
```

※ Example scripts occupies four nodes (select=4) with 32 processors per node (total 128 processors)

■ Batch script to run a hybrid MPI (with IntelMPI)/OpenMP job (hybrid_intelmpi.sh)

```
#!/bin/sh
#PBS -N hybrid_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=4:ncpus=32:mpiprocs=2:ompthreads=16
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

mpirun ./a.out
```

※ Example scripts occupies four nodes (select=4) with two MPI processes and 16 threads per node (total 8 MPI processors and 64 OpenMP threads).

■ Batch script to run a hybrid MPI (with OpenMPI)/OpenMP job (hybrid_openmpi.sh)

```
#!/bin/sh
#PBS -N hybrid_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=4:ncpus=32:mpiprocs=2:ompthreads=16
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

mpirun --map-by NUMA:PE=16 ./a.out
```

※ Example scripts occupies four nodes (select=4) with two MPI processes and 16 threads per node (total 8 MPI processors and 64 OpenMP threads).

■ Batch script to run a hybrid MPI (with Mvapich2)/OpenMP job (hybrid_mvapich2.sh)

```
#!/bin/sh
#PBS -N hybrid_job
#PBS -V
#PBS -q normal
#PBS -A {PBS option name}   # Please refer to “PBS option name table”
#PBS -l select=4:ncpus=32:mpiprocs=2:ompthreads=16
#PBS -l walltime=04:00:00

cd $PBS_O_WORKDIR

TOTAL_CPUS=$(wc -l $PBS_NODEFILE | awk '{print $1}')

mpirun_rsh -np ${TOTAL_CPUS} -hostfile $PBS_NODEFILE
OMP_NUM_THREADS=$OMP_NUM_THREADS ./a.out
```

※ Example scripts occupies four nodes (select=4) with two MPI processes and 16 threads per node (total 8 MPI processors and 64 OpenMP threads).

■ Batch script to run a commercial SW “Gaussian”

```
#!/bin/sh
#PBS -N gauss_job
#PBS -V
#PBS -q norm_skl
#PBS -l select=1:ncpus=40:mpiprocs=1:ompthreads=40:mem=4gb
#PBS -l walltime=04:00:00
#PBS -A gaussian

cd $PBS_O_WORKDIR

export g16root="/apps/commercial/G16"
export g16BASIS=${g16root}/g16/basis
export GAUSS_EXEDIR=${g16root}/g16
export GAUSS_LIBDIR=${g16root}/g16
export GAUSS_ARCHDIR=${g16root}/g16/arch
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:${g16root}/g16"
export PATH="${PATH}:${g16root}/g16"
export GAUSS_SCRDIR="/scratch/${USER}"


g16 test.com
```

※ Users can set the maximum memory use like the example above (mem=4gb), but the job can be killed if the job uses more than the assigned memory limit.

■ Batch script to run tensorflow within singularity container (single node-KNL)

```
#!/bin/sh
#PBS -N tensorflow_job
#PBS -V
#PBS -q normal
#PBS -A tf
#PBS -l select=1:ncpus=68:ompthreads=32
#PBS -l walltime=1:00:00

cd $PBS_O_WORKDIR
module load singularity/3.0.1
export OMP_NUM_THREADS=32

singularity exec tensorflow-1.12.0-py3.simg python convolutional.py
```

※ The paths of the example files:

/apps/applications/tensorflow/1.12.0/tensorflow-1.12.0-py3.simg

/apps/applications/tensorflow/1.12.0/examples/convolutional.py

- We recommend users to first copy those files to the directory where user will execute them.

■ Batch script to run tensorflow within singularity container (multiple nodes-KNL)

```
#!/bin/sh
#PBS -N tensorflow-horovod_job
#PBS -V
#PBS -q normal
#PBS -A tf
#PBS -l select=100:ncpus=68:mpiprocs=2:ompthreads=32
#PBS -l walltime=10:00:00

cd $PBS_O_WORKDIR
export batch_size=128
export MODEL=resnet50
export inter_op=2
export intra_op=32
export OMP_NUM_THREADS=32
export python_script=tf_cnn_benchmarks.py
module load gcc/6.1.0 openmpi/3.1.0 singularity/2.5.2

mpirun singularity exec tensorflow-1.12.0-py3.simg \
python  $python_script --mkl=True --forward_only=False \
--num_batches=200 --kmp_blocktime=0 --num_warmup_batches=50 \
--num_inter_threads=$inter_op --distortions=False --optimizer=sgd \
--batch_size=$batch_size --num_intra_threads=$intra_op --data_format=NCHW \
--model=$MODEL --variable_update=horovod --horovod_device=cpu
```

※ The above script is an example of the execution of a distributed Tensorflow on a multiple nodes using a horovod(OpenMPI).

- Example files can be found /apps/applications/tensorflow/1.12.0/benchmarks/scripts/tf_cnn_benchmarks/, and users need to copy those data into where those files will be executed.
- In the example, two MPI tasks are created for each KNL nodes and 32 threads per task are used.
- Make sure to set the OMP_NUM_THREADS environment variable so that too many threads are created to cause resource shortage errors.
- To optimize performance, set the following parameters appropriately: --batch_size, --num_inter_threads (inter_op, the maximum possible number of parallel excutable operators) --num_intra_threads (intra_op, the maximum number of parallel threads used to run the operator), OMP_NUM_THREADS (=intra_op, the number of OpenMP parallel threads).
- Note) num_inter_threads × num_intra_threads ≤ num_total_threads
  - num_total_threads is 68 and 40 for KNL and SKL node respectively (hyperthreading off)

○ **E-mail alert**

```
$ qsub -m <options> -M <user_e_mail_address>

ex) qsub -m abe -M abc@def.com hello_world.sh
```

options	description
a	send mail when job is aborted (default).
b	send mail when job begins.
e	send mail when job terminates.
n	no mail.

3) Submitting interactive jobs

○ Use option “-I” instead of batch script.

```
$ qsub -I -l select=1:ncpus=64:ompthreads=1 -l walltime=24:00:00 \
  -q normal  -A {PBS option name}
```

※ Please refer to “PBS option name table” in the previous section.

○ To use graphic environment when submitting interactive job(s), use option “-X”.

```
$ qsub -I -X -l select=1:ncpus=64:ompthreads=1 -l walltime=24:00:00 \
  -q normal -A {PBS option name}
```

○ To inherit existing environment variables when submitting interactive jobs, use option “-V”

```
$ qsub -I -V -l select=1:ncpus=64:ompthreads=1 -l walltime=24:00:00 \
   -q normal -A {PBS option name}
```

※ Since a debugging node is not given, debugging with interactive job submission is recommended.

■ Example of running tensorflow on a compute node with interactive job submission

```
$ qsub -I -V –l select=1:ncpus=68:ompthreads=32 \
  -l walltime=24:00:00 -q normal -A tf

$ export OMP_NUM_THREADS=32;
singularity exec tensorflow-1.12.0-py3.simg python convolutional.py
```

※ The paths of the example files:
/apps/applications/tensorflow/1.12.0/tensorflow-1.12.0-py3.simg
/apps/applications/tensorflow/1.12.0/examples/convolutional.py

- We recommend users to first copy those files to the directory where user will execute them.

4) Monitoring jobs

○ Display information about jobs

```
$ showq
```

○ Query status of compute nodes

```
$ pbs_status
```

or

```
$ pbsnodes -ajS
```

column	description
mem f/t	memory (unit: GB)
ncpus f/t	available CPU processors

※ Here, *f* indicates the number of *free* mem/ncpus while *t* indicates the number of *total* mem/ncpus.

○ Query of job status

```
$ qstat <-a, -n, -s, -H, -x, ... >

  ex> qstat
 Job id           Name            User           Time Use S Queue
 -------------------------------------------------------------------
 0001.pbcm        test_01         user01         8245:43: R normal
 0002.pbcm        test_02         user02         8245:44: R flat
 0003.pbcm        test_03         user03         7078:45: R norm_skl
 0003.pbcm        test_04         user04         1983:11: Q long
```

○ Query of all types of job status information

```
  $ qstat -f <job ID>

  ex> qstat -f 0000.pbcm
 Job Id: 0000.pbcm
     Job_Name = test
     Job_Owner = user0@pbcm.cm.cluster
     resources_used.cpupercent = 6416
     resources_used.cput = 8245:43:20
     resources_used.mem = 33154824kb
     resources_used.ncpus = 64
     resources_used.vmem = 99989940kb
     resources_used.walltime = 128:54:21
     job_state = R
  <...omitting...>
```

○ Query of estimated start time of my job(s)

```
$ qstat -i -w -T -u <user ID>
```

(example)

```
$ qstat -i -w -T -u user01
pbs:
                                                                          Est
                                                       Req'd    Req'd     Start
Job ID         Username   Queue   Jobname    SessID  NDS  TSK  Memory   Time     S  Time
-------------- ---------- ------- ---------- ------- ---- ---- ------   -----    -  -----
0000001.pbs    user01     long    test1      --         1   32 --       48:00:00 Q  Today    11:38
0000003.pbs    user01     flat    test3      --         1   32 --       48:00:00 Q  Today    11:39
0000004.pbs    user01     flat    test4      --         1   32 --       48:00:00 Q  Today    11:39
```

5) Managing jobs

- ○ Delete job(s)

```
$ qdel <job ID>
```

- ○ Suspend/resume job(s)

```
$ qsig -s <suspend/resume> <job ID>
```

3. Others

a. How to use Singularity container image

○ Load a singularity module

```
$ module load singularity/2.5.2 (or 3.0.1 or 2.5.1 or 2.4.1)
              or
$ $HOME/.bash_profile
  export PATH=$PATH:/opt/singularity/2.5.2/bin
```

○ Execute a shall within Singularity container

```
$ singularity shell [name of image]

$ singularity shell tensorflow-1.12.0-py3.simg
Singularity: Invoking an interactive shell within container...

Singularity tensorflow-1.12.0-py3.simg:tensorflow>
```

○ Execute user's program within Singularity container

```
$ singularity exec [name of image] execution_command

$ singularity exec tensorflow-1.12.0-py3.simg python convolutional.py
```

Software (Framework)	Container image file
tensorflow 1.12.0	/apps/applications/tensorflow/1.12.0/tensorflow-1.12.0-py3.simg

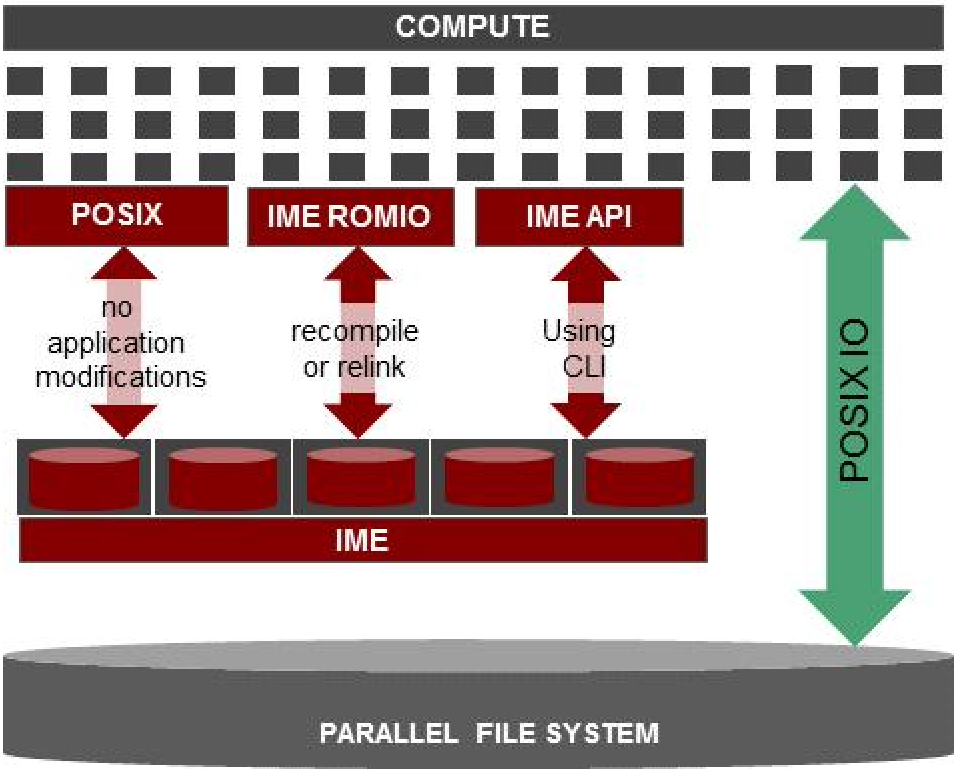
※ Users are recommended to copy the container image file to users' work directory for use.

※ Users can copy a convolutional model example (convolutional.py) and "data" directory from the directory /apps/applications/tensorflow/1.12.0/examples to users' work directory and test them.

b. How to use Burst Buffer (BB)

○ Concept

The burst buffer IME acts as a cache for the NURION file system of /scratch. The data access method through IME is shown below.



The IME is mounted on the client nodes, that compute node assigned to the burst_buffer queue, using the user-level file system FUSE (File System In USErspace). Note that the IME acts as a cache for /scratch, so the file system must be mounted beforehand. The IME directory location is /scratch_ime, which means that when user first access the directory(/scratch_ime/$USER), user can see the structure of all directories and files in the scratch(/scratch/$USER) file system the same. These are not present in the actual IME device and will be performed by caching the data from /scratch to the IME when using the burst buffer. There are two main methods of executing an application using the IME.

① The first is to use the IME mount point of /scratch_ime as the I/O directory and it can perform the existing POSIX based I/O without any compilation. That is, user only need to set the I/O directory under /scratch_ime/$USER_ID/ while submitting a job through burst_buffer queue.

ex) INPUT="/scratch_ime/$USER/input.dat", OUTPUT="/scratch_ime/$USER/output.dat"

② The second is to use the mvapich2/2.3.1 module, which supports MPI-IO. The application must be recompiled using the loaded IME-based MPI library. In addition, a path of files or directories should be defined with IME protocol as follows:

ex) OUTFILE=ime:///scratch/$USER/output.dat (Also, please see the example of job script below)

※ Compilers supporting mvapich2/2.3.1

: gcc/6.1.0, gcc/7.2.0, intel/17.0.5, intel/18.0.1, intel/18.0.3, intel/19.0.4, pgi/18.10

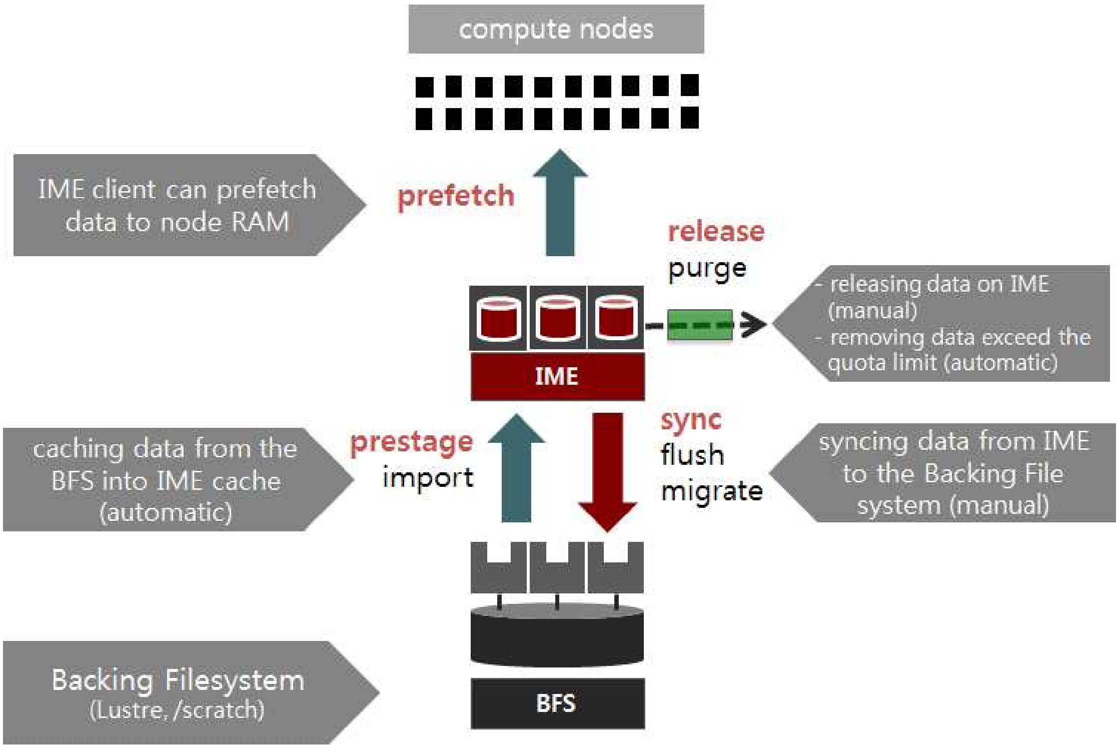
■ **Example of job script: How to use ime mpi (Mvapich2/2.3.1)**

```
$ module load mvapich2/2.3.1
```

```
#!/bin/sh
#PBS -N mvapich2_ime
#PBS -V
#PBS -q normal
#PBS -P burst_buffer
#PBS -A {PBS option name}   # Please refer to the PBS name table
#PBS -l select=2:ncpus=16:mpiprocs=16
#PBS -l walltime=05:00:00

cd $PBS_O_WORKDIR
TOTAL_CPUS=$(wc -l $PBS_NODEFILE | awk '{print $1}')
OUTFILE=ime:///scratch/$USER/output.dat
mpirun_rsh -np ${TOTAL_CPUS} -hostfile $PBS_NODEFILE ./a.out > $OUTFILE
```

To run the application through the IME, you have to understand the life cycle of the data shown in the figure below. The IME data processing phase consists of Prestage, Prefetch, Sync, and Release phases, and IME-API (#ime-ctl) is provided for each.


compute nodes
IME client can prefetch data to node RAM
prefetch
release
purge
- releasing data on IME (manual)
- removing data exceed the quota limit (automatic)
IME
caching data from the BFS into IME cache (automatic)
prestage
import
sync
flush
migrate
syncing data from IME to the Backing File system (manual)
Backing Filesystem
(Lustre, /scratch)
BFS

[Life cycle of data]

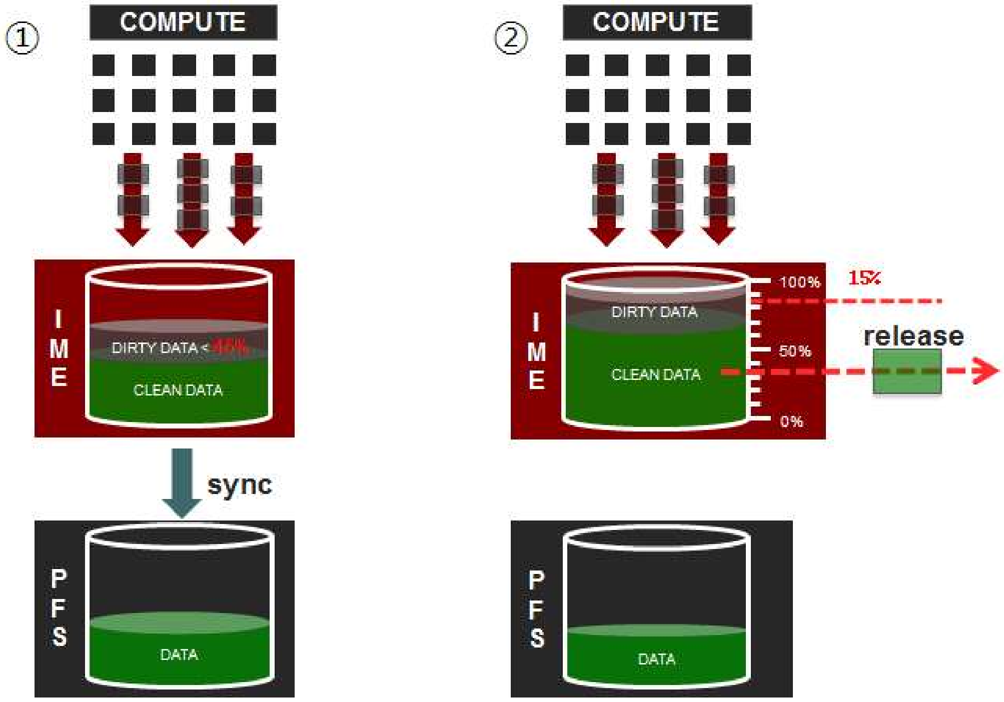
ime-ctl -i $INPUT_FILE	Execute stage-in for job data into IME (caching the data from /scratch to /scratch_ime)
ime-ctl -r $OUTPUT_FILE	Synchronize IME data with the parallel file system (Sending the data from /scratch_ime to /scratch)
ime-ctl -p $TMP_FILE	Clear data from IME (Purge the data of /scratch_ime)
ime-ctl -s $FILE	Providing status information for IME data

* #ime-ctl --help for detailed options

○ Data processing

The total capacity of IME is about 900TB, and it is automatically flushed or deleted in the file system(/scratch) depending on usage. IME automatically allocates cache space according to the following two threshold settings.

① When the total capacity of newly created data(Dirty Data) is 45% or more
② When the total remaining free space is less than 15%



○ Notes

The IME is burdened with the task of caching the PFS data to the IME device and flushing or synchronizing the cache data back to the PFS when performing the initial operation. Therefore, it can be expected to improve performance in applications that have a large amount of small I/O, frequent checkpoints, or long I/O execution time compared to PFS (Luster).

Also, the IME (about 0.9PB) is relatively small in capacity because it is used for the cache of PFS (about 20PB). Therefore, IME try to maintain the available capacity below the specified threshold by automatically sync-ing or flushing of the users' data without any notices in advance.

※ Deletion of cached data in the IME **must** use the given IME-API command. For example, in /scratch_ime, the data sored in the actual /scratch will be deleted if one uses "rm" command instead of IME-API command.

c. How to use Flat node

○ On “Flat nodes” that user can use by submitting job(s) with “flat” queue, the MCDRAM (16GB) is configured entirely as addressable memory. When using Flat node, user can utilize max 102 GB memory per node.

○ For using “flat node”, the “numactl” command should be used to specify a preferred or default memory domain. For example, to allocate all your memory out of the MCDRAM, you sould launch your executable with the following:

```
$ qsub <pbs options> numactl -m 1 my_app.x
```

○ The above will fail if your applications requires more than 16 GB of memory. To instead perfer MCDRAM for your first 16GB, but allocated remaining data in the DDR memory, use the “-p” flag to numactl.

```
$ qsub <pbs options> numactl -p 1 my_app.x
```

Note that user must submit job(s) though “flat” queue for using Flat node. (PBS option: -q flat)

4. User Support

Please contact the Helpdesk if user has any problem or question.

Type	Website
Technical support **(Contract/account/system)**	**https://helpdesk.ksc.re.kr**
Education	https://kacademy.kisti.re.kr
Technical support (**optimization/parallization)**	https://enables.ksc.re.kr