뉴론(NEURON)
지침서

2024. 04.

목 차

Ⅰ. 시스템

Ⅰ-1. 시스템 개요 및 구성

가. 시스템 운영 목적

- 5호기 시스템이 Knight Landing 기반의 시스템으로 결정됨으로 GPU 기반의 시스템 운영을 통한 사용자의 다양한 수요 대응
- 대형 메모리 필요 작업 수요 충족을 위한 대용량 메모리 노드 운영
- '19.5 서비스 개시, 5호기와 파일시스템 공유, 차세대 신기술(FPGA, AMD EPYC, Optane 등) 지속적으로 채택/확장

나. 시스템 구성

최신 시스템 구성, 계산노드 사양, 스토리지 구성의 상세 정보는 아래 국가슈퍼컴퓨팅센터 > 보유자원 > 뉴론 웹 페이지에서 확인 가능

https://www.ksc.re.kr/byjw/neuron

Ⅰ-2. 사용자 환경

가. 계정발급

1. 뉴론(NEURON) 시스템의 사용을 승인받은 연구자는 KISTI 홈페이지 (https://www.ksc.re.kr) 웹 서비스를 통해 계정을 신청한다.

1) 신청 방법 :

KISTI 홈페이지 웹 사이트 접속, (상단) 사용신청 -> (상단) 신청 -> 신청서 선택

- 무료 계정 : 누리온 시스템 혁신지원 프로그램, 초보 사용자
- 유료 계정 : 일반사용자, 학생 사용자
- 계정 발급 완료 시 신청서에 기입한 이메일로 계정 관련 정보 발송

2) OTP (One Time Password, 일회용 비밀번호) 인증코드 발급

- **수신하신 계정 정보 이메일을 참고하여** 아래와 같이 작성하여 account@ksc.re.kr을 통해 인증코드를 발급받는다.

메일 제목	OTP 인증코드 발송 요청 - 사용자 ID (예) OTP 인증코드 발송 요청 - x123abc
수신인	account@ksc.re.kr
메일 내용(예제)	로그인 ID: x123abc 휴대폰 번호: 010-1234-5678 이름: 홍길동 통신사: LG 유플러스(or SKT / KT)

3) OTP 앱 설치

- 슈퍼컴퓨팅 보안 접속을 위해 OTP 스마트폰 앱이 제공된다.
- OTP 스마트폰 앱은 안드로이드 앱 스토어(Google Play)나 아이폰 앱 스토어(App Store)에서 “Any OTP”로 검색 후 미래 기술(mirae-tech)에서 개발한 앱을 설치하여 사용할 수 있다.
- 슈퍼컴퓨터 로그인 시 “Any OTP” 앱의 OTP 보안 숫자를 반드시 입력해야 한다.

※ 스마트폰을 사용하고 있지 않은 사용자의 경우, 계정담당자(account@ksc.re.kr)에게 문의

※ 자세한 OTP 설치 및 이용 방법은 KISTI 홈페이지 > 기술지원 > 지침서에서 “OTP 사용자 매뉴얼” 참조

※ LG 유플러스의 경우에는 문자가 스팸 처리되므로 이메일로 안내

나. 로그인

- 사용자는 뉴론 시스템 로그인 노드(neuron01.ksc.re.kr, neuron02.ksc.re.kr, neuron03.ksc.re.kr)를 통해서 접근이 가능하다. (하단, 노드 구성 참조)

※ 웹 브라우저를 통해 MyKSC(KISTI 슈퍼컴퓨터 웹 서비스 포털, https://my.ksc.re.kr)에 로그인하여 GUI 기반의 HPC 및 AI/데이터분석 서비스를 활용할 수 있음 (사용법은 MyKSC 지침서 참조)

- 기본 문자셋(encoding)은 유니코드(UTF-8)이다.
- 로그인 노드에 대한 접근은 ssh, scp, sftp, X11만 허용된다.

1. 유닉스 또는 리눅스 환경

```
$ ssh -l <사용자ID> neuron01.ksc.re.kr -p 22
```

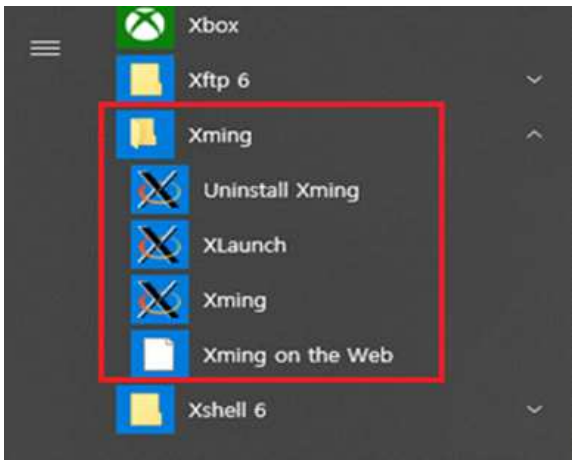
- [-p 22]: 포트 번호 명시를 위한 것으로 생략 가능
- X환경 실행을 위해 XQuartx 터미널을 이용

※ 프로그램은 인터넷을 통해 무료로 다운로드 후 설치

2. 윈도우 환경

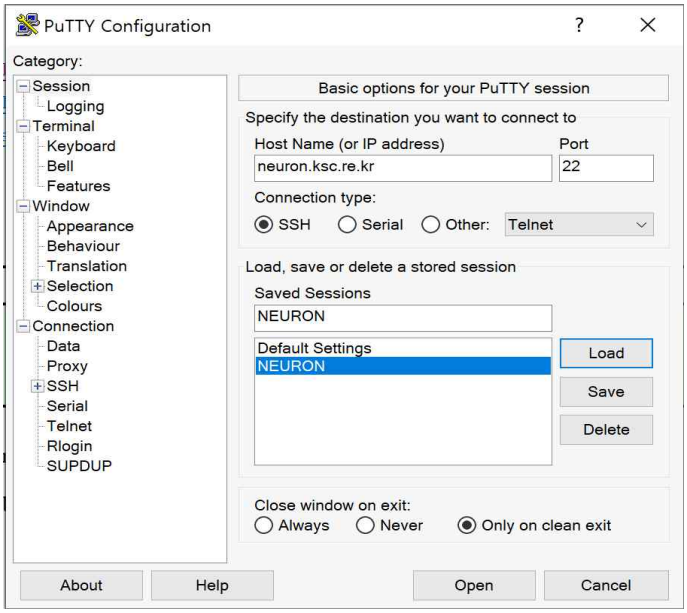
- X환경 실행을 위해 Xming 실행

※ 프로그램은 인터넷을 통해 무료로 다운로드 후 설치

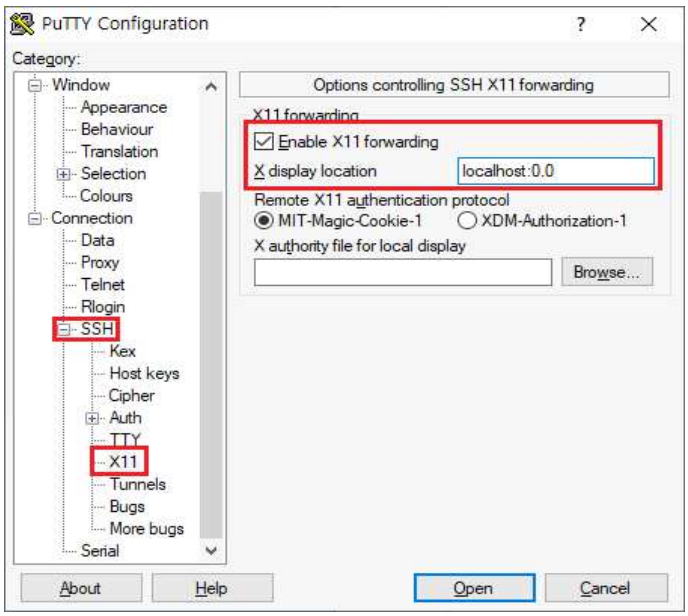


- putty, Mobaxterm, 또는 SSH Secure Shell Client 등의 ssh 접속 프로그램을 이용
  - Host Name : neuron.ksc.re.kr, Port : 22, Connection type : SSH

※ 프로그램은 인터넷을 통해 무료로 다운로드 가능



- ssh -> X11 tap -> check “Enable X11 forwarding”
- X display location : localhost:0.0



※ 만약, DNS 캐싱 문제로 접속이 안 될 경우에는 캐시를 정리 (명령 프롬프트에서 ipconfig /flushdns 명령어 수행)하고 재접속

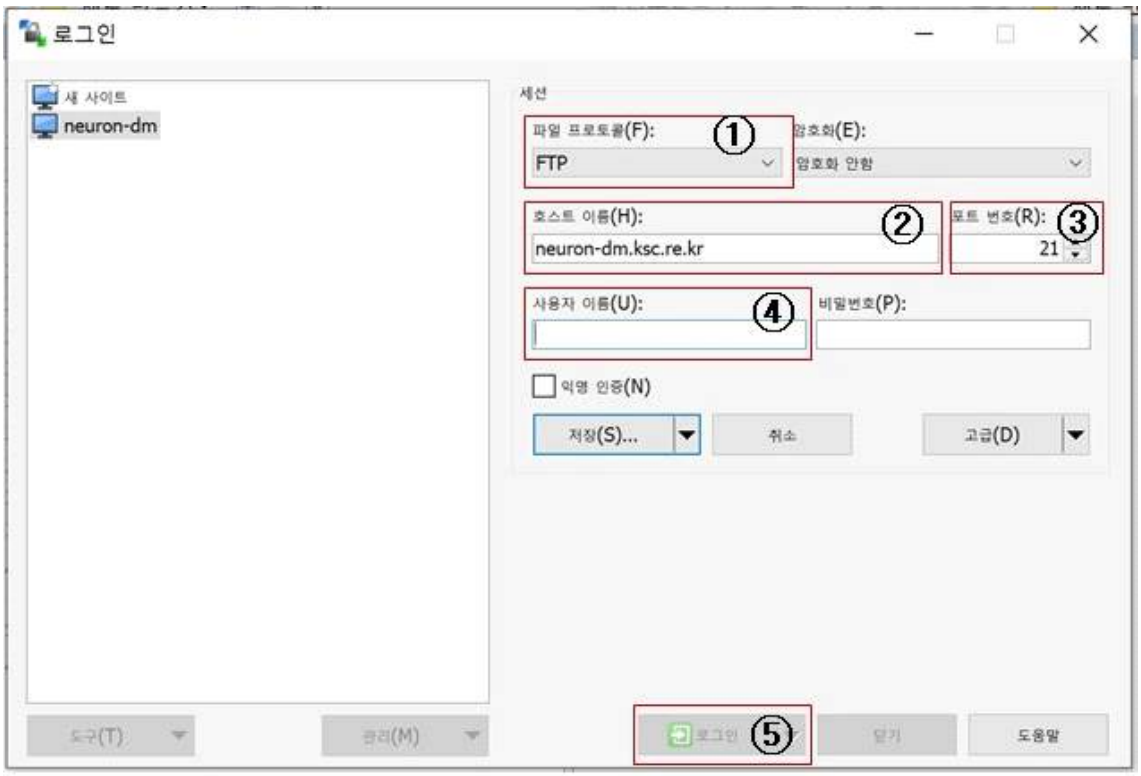
```
C:₩> ipconfig /flushdns
```

3. 파일 송수신

- FTP 클라이언트를 통해 ftp(OTP 없이 접속 가능)나 sftp로 접속하여 파일을 송수신한다.

```
$ ftp neuron-dm.ksc.re.kr
또는
$ sftp [사용자ID@]neuron-dm.ksc.re.kr [-P 22]
```

- 윈도우 환경에서는 WinSCP와 같이 무료로 배포되고 있는 FTP/SFTP 클라이언트 프로그램을 이용하여 접속한다.



- FTP(File Tranfer Protocal)을 이용하며, OTP를 입력하지 않고 파일 전송 가능
- SFTP(Secure-FTP)을 이용하며, 파일 전송 시 OTP를 입력해야 함. (FTP보다 안전한 전송방식)

4. 노드 구성

노드	호스트명	CPU Limit	비고
로그인 노드	neuron.ksc.re.kr (neuron01.ksc.re.kr, neuron02.ksc.re.kr, neuron03.ksc.re.kr)	120분	• ssh/scp 접속 가능 • 컴파일 및 batch 작업제출용 • ftp/sftp 접속 불가
Datamover 노드	neuron-dm.ksc.re.kr	-	• ssh/scp/sftp/ftp 접속 가능 • 컴파일 및 작업제출 불가

- 로그인 노드 상세 사양

1) AMD 노드

- 호스트명 : neuron01.ksc.re.kr, neuron03.ksc.re.kr
- CPU : AMD EPYC 7543 2socket
- 메모리 : 512GB
- GPU : NVIDIA A100 2ea (GPU당 각각 7개씩 총 14개의 인스턴스로 구성)

2) Intel 노드

- 호스트명 : neuron02.ksc.re.kr
- CPU : Intel Xeon Gold 5217 2socket
- 메모리 : 384GB
- GPU : NVIDIA V100 1ea

※ DM 노드에는 Nurion의 홈디렉터리(/nurion_home01/[userid])와 스크래치디렉터리(/nurion_scratch/[userid])가 마운트되어 있어서 두 시스템에서 동일 userid를 사용할 경우 시스템 간 파일 이동이 가능함.

※ wget, git을 이용한 다운로드 및 대용량 데이터의 전송은 CPU Limit이 없는 Datamover 노드를 사용할 것을 권장함 (로그인 노드에서 수행 시에 CPU Limit에 따라 전송 중에 끊어질 수 있음)

5. 디버깅 노드 제공

- 디버깅, 컴파일, 수행코드 테스트 등의 목적으로 2개의 GPU 노드 제공(실제 작업 수행은 제한함)
- 각 노드당 2개의 CPU(Xeon2.9GHz/32Cores)와 2개의 GPU(V100)가 장착되어 있음
- 로그인 노드(glogin[01.02])에서 ssh로 직접 접속 가능하며(ssh gdebug01 또는 gdebug02), 스케줄러를 통한 서비스는 더 이상 지원하지 않음

다. 사용자 쉘(shell) 변경

- 뉴론 시스템의 로그인 노드는 기본 쉘로 bash이 제공된다. 다른 쉘로 변경하고자 할 경우 chsh 명령어를 사용한다.

```
$ chsh
```

- 현재 사용 중인 쉘을 확인하기 위해서 echo $SHELL을 이용하여 확인한다.

```
$ echo $SHELL
```

- 쉘의 환경설정은 사용자의 홈 디렉터리에 있는 환경설정 파일(.bashrc, .cshrc 등)을 수정하여 사용하면 된다.

라. 사용자 비밀번호 변경

- 사용자 패스워드를 변경하기 위해서는 로그인 노드에서 passwd 명령을 사용한다.

```
$ chsh
```

※ 패스워드 관련 보안 정책

- 사용자 패스워드 길이는 최소 9자이며, 영문, 숫자, 특수문자의 조합으로 이뤄져야 한다. 영문 사전 단어는 사용이 불가하다.
- 사용자 패스워드 변경 기간은 2개월로 설정(60일)된다.
- 새로운 패스워드는 최근 5개의 패스워드와 유사한 것을 사용할 수 없다.
- 최대 로그인 실패 허용 횟수 : 5회
  - 5회 이상 틀릴 경우, 이 계정의 ID는 lock이 걸리므로, 계정담당자(acccount@ksc.re.kr)에게 문의해야 한다.
  - 같은 PC에서 접속을 시도하여 5회 이상 틀릴 경우, 해당 PC의 IP 주소는 일시적으로 block 되므로 이 경우에도 계정담당자(account@ksc.re.kr)에게 문의해야 한다.
- OTP 인증 오류 허용 횟수 : 5회
  - 5회 이상 틀릴 경우, 계정담당자(account@ksc.re.kr)에게 문의해야 한다.

마. 제공 시간

큐 명 (CPU종류_GPU종류_GPU개수)	노드명	구좌당 노드 제공 시간	1노드 1시간 사용요금
cas_v100nv_8	gpu[01-05]	400	2,499원
cas_v100nv_4	gpu[06-09]	700	1,426원
cas_v100nv_4	gpu[10-24]	700	1,426원
cas_v100_2	gpu[25-29]	1,200	831원
amd_a100nv_8	gpu[30-35]	300	3,334원
amd_a100_4	gpu[36-37]	500	1,995원
amd_a100_2	gpu[38-39]	800	1,249원
skl	skl[01-10]	5,400	184원
bigmem	bigmem01	9,700	104원
	bigmem02	2,800	353원
	bigmem03	1,200	833원

※ 공유 노드 정책으로 인하여 작업이 사용한 core, gpu 개수만큼 과금 부과

※ 최신 요금 정보는 요금 계산기 (https://www.ksc.re.kr/jwjg/gjbg/yggsg) 참고

바. 작업 디렉터리 및 쿼터 정책

- 홈 디렉터리 및 스크래치 디렉터리에 대한 정보는 아래와 같다.

구분	디렉터리 경로	용량 제한	파일 수 제한	파일 삭제 정책	백업 유무
홈 디렉터리	/home01	64GB	100K	-	X
스크래치 디렉터리	/scratch	100TB	4M	15일 동안 접근하지 않은 파일은 자동 삭제함	X

*** NEURON 시스템은 백업을 지원하지 않음.**

- 홈 디렉터리는 용량 및 I/O 성능이 제한되어 있기 때문에, 모든 계산 작업은 스크래치 디렉터리인 /scratch의 사용자 작업 공간에서 이루어져야 한다.
- 사용자 디렉터리 용량 확인(login 노드에서 실행)

```
$ quotainfo
```

- 파일 삭제 정책은 파일 속성 중 atime을 기준으로 적용된다.
  - atime(Access timestamp): 마지막 접근 시간을 나타냄
  - mtime(Modified timestamp): 파일의 마지막 변경 시간 의미
  - ctime(Change timestamp):파일의 퍼미션, 소유자, 링크 등에 의한 inode 변경 시간 의미

※ 삭제 정책(15일 동안 접근하지 않은 파일은 자동 삭제함)에 따라 삭제된 파일의 복구는 기본적으로 불가능하나 파일 관리를 잊은 사용자에게 유예기간을 제공하기 위해 퍼지 정책을 일부 변경합니다. (2023.05. 시행)

- 삭제 정책 수행 후 대상이 되는 파일은 앞에 접두사(ToBeDelete_)가 붙게 됩니다.
  예) ToBeDelete_file1.txt, ToBeDelete_file2.txt
  - 대상 파일이 필요한 경우 사용자는 직접 접두사가 붙은 파일명을 복구시키고 atime을 업데이트해야 합니다.
- 일정 기간(20일~30일) 후 ToBeDelete_붙은 파일은 일괄 삭제됩니다. 이후 해당 파일은 복구가 불가능합니다.
- atime은 stat, ls -lu 명령 등으로 확인할 수 있다.

```
[login01 ~]# stat file1
File: ‘file1’
Size: 0 Blocks: 0 IO Block: 4096 regular empty file
Device: 805h/2053d Inode: 3221237903 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)
Access: 2022-04-05 08:52:30.113048319 +0900
Modify: 2022-04-05 08:52:30.113048319 +0900
Change: 2022-04-05 08:52:30.113048319 +0900
```

```
$ ls -lu test.file
-rw-r--r-- 1 testuser testgroup 58 Jan 1 17:06  test.file
```

Ⅰ-3. 사용자 프로그래밍 환경

가. 프로그래밍 도구 설치 현황

구분		항목
컴파일러		• gcc/4.8.5 • gcc/10.2.0 • intel/19.1.2 • nvidia_hpc_sdk/22.7
컴파일러 의존 라이브러리		• hdf4/4.2.13 • hdf5/1.10.2 • lapack/3.7.0 • ncl/6.5.0 • netcdf/4.6.1
CUDA 라이브러리		• cuda/11.4
CUDA 라이브러리 (cuda 라이브러리만 설치된 버전)		• cuda/10.0 • cuda/10.2 • cuda/11.0 • cuda/11.1 • cuda/11.2 • cuda/11.3 • cuda/11.5 • cuda/11.6
MPI 라이브러리	CUDA	• mvapich2-2.3.6 • cudampi/openmpi-3.1.5 • cudampi/openmpi-4.1.1
	None CUDA	• mpi/impi-19.1.2 • mpi/mvapich2-2.3.6 • mpi/openmpi-3.1.5 • mpi/openmpi-4.1.1
MPI 의존 라이브러리		• fftw_mpi/2.1.5 • fftw_mpi/3.3.7
상용 소프트웨어		• gaussian/g16 • gaussian/g16.b01 • gaussian/g16.c01
응용 소프트웨어		• cmake/3.16.9 • python/3.7.1 • python/3.9.5 • namd/2.14

	• qe/7.0_v100 • qe/7.0_a100 • gromacs/2021.3 • lammps/27Oct2021_v100 • lammps/27Oct2021_a100 • R/3.5.0 • git/2.9.3 • git/2.35.1 • htop/3.0.5 • nvtop/1.1.0
가상화 모듈	• singularity/3.6.4 • singularity/3.9.7
기계학습 프레임워크 소프트웨어	• conda/pytorch_1.11.0 • conda/tensorflow_2.4.1
NGC(Nvidia GPU Cloud) 컨테이너 기반 소프트웨어	• ngc/caffe:20.03-py3 • ngc/gromacs:2020.2 • ngc/lammps:29Oct2020 • ngc/paraview:5.9.0-py3 • ngc/pytorch:22.03-py3 • ngc/qe:6.7 • ngc/tensorflow:22.03-tf1-py3 • ngc/tensorflow:22.03-tf2-py3

- conda/pytorch_1.11.0, conda/tensorflow_2.4.1 module은 pytorch/tensorflow 사용을 위한 라이브러리가 설치되어 있는 module로, conda 명령을 사용하려면 python/3.7.1 module 적용 후 conda 명령을 사용해야 한다.
- Neuron 시스템에서의 인공지능 프레임워크는 anaconda 환경을 사용하는 것을 권장함(라이선스 조건 확인)
- 사용자 요구 기반의 Singularity 컨테이너 이미지를 빌드 및 구동하여 사용자 프로그램을 실행할 수 있음
  - ◦ [별첨3] Singularity 컨테이너 사용법'을 참조

※ GPU가 장착되지 않은 노드를 사용하기 위해서는 None CUDA MPI 라이브러리를 사용해야 한다.

나. 컴파일러 사용법

1. 컴파일러 및 MPI 환경설정(modules)

1) 모듈 관련 기본 명령어

※ 사용자 편의를 위해 "module" 명령어는 "ml" 명령어로 축약하여 사용할 수 있음

• 사용 가능한 모듈 목록 출력

사용할 수 있는 컴파일러, 라이브러리 등의 모듈 목록을 확인할 수 있다.

```
$ module avail
혹은
$ module av
```

• 사용할 모듈 추가

사용하고자 하는 컴파일러, 라이브러리 등의 모듈을 추가할 수 있다.
사용할 모듈들을 한 번에 추가할 수 있다.

```
$ module load [module name] [module name] [module name] ...
혹은
$ module add [module name] [module name] [module name] ...
혹은
$ ml [module name] [module name] [module name] ...
ex) ml gcc/4.8.5 singularity/3.9.7
```

• 사용 모듈 삭제

필요 없는 모듈을 제거한다. 이때 한 번에 여러 개의 모듈을 삭제할 수 있다.

```
$ module unload [module name] [module name] [module name] ...
혹은
$ module rm [module name] [module name] [module name] ...
혹은
$ ml -[module name] -[module name] -[module name] ...
ex) ml -gcc/4.8.5 -singularity/3.9.7
```

• 사용 모듈 목록 출력

현재 설정된 모듈 목록을 확인할 수 있다.

```
$ module list
혹은
$ module li
혹은
$ ml
```

• 전체 사용 모듈 일괄 삭제

```
$ module purge
```

• 모듈 설치 경로 확인

```
$ module show [module name]
```

• 모듈 찾기

```
$ module spider  [module | string | name/version ]
```

• 사용자 모듈 모음(collection) 저장 관리

```
# 현재 로드된 모듈들을 default 모듈 모음에 저장하며, 다음 로그인 시 자동 로드됨
$ module save
# 현재 로드된 모듈들을 지정된 이름을 가진 사용자 모듈 모음으로 저장함
$ module save [name]
# 사용자 모듈 모음을 로드함
$ module restore [name]
# 사용자 모듈 모음의 내용을 출력함
$ module describe [name]
# 사용자 모듈 모음 리스트를 출력함
$ module savelist
# 사용자 모듈 모음을 삭제함
$ module disable [name]
```

2. 순차 프로그램 컴파일

순차 프로그램은 병렬 프로그램 환경을 고려하지 않은 프로그램을 말한다. 즉, OpenMP, MPI와 같은 병렬 프로그램 인터페이스를 사용하지 않는 프로그램으로써, 하나의 노드에서 하나의 프로세서만 사용해 실행되는 프로그램이다. 순차 프로그램 컴파일 시 사용되는 컴파일러별 옵션은 병렬 프로그램을 컴파일 할 때도 그대로 사용되므로, 순차 프로그램에 관심이 없다 하더라도 참조하는 것이 좋다.

1) Intel 컴파일러

Intel 컴파일러를 사용하기 위해서 필요한 버전의 Intel 컴파일러 모듈을 추가하여 사용한다. 사용 가능한 모듈은 module avail로 확인할 수 있다.

```
$ module load intel/18.0.2
```

※ 프로그래밍 도구 설치 현황표를 참고하여 사용 가능 버전 확인

• 컴파일러 종류

컴파일러	프로그램	소스 확장자
icc / icpc	C / C++	.C, .cc, .cpp, .cxx,.c++
ifort	F77/F90	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FTN, .FPP, .F90

• Intel 컴파일러 사용 예제

다음은 test 예제파일을 intel 컴파일러로 컴파일하여 실행파일 test.exe를 만드는 예시임

```
$ module load intel/19.1.2
$ icc -o test.exe test.c
혹은
$ ifort -o test.exe test.f90
$ ./test.exe
```

※ /apps/shell/job_examples에서 작업제출 test 예제파일을 복사하여 사용 가능

2) GNU 컴파일러

GNU 컴파일러를 사용하기 위해서 필요한 버전의 GNU 컴파일러 모듈을 추가하여 사용한다. 사용 가능한 모듈은 module avail로 확인할 수 있다.

```
$ module load gcc/10.2.0
```

※ 프로그래밍 도구 설치 현황표를 참고하여 사용 가능 버전 확인

※ 반드시 "gcc/4.8.5" 이상 버전을 사용

• 컴파일러 종류

컴파일러	프로그램	소스 확장자
gcc / g++	C / C++	.C, .cc, .cpp, .cxx,.c++
gfortran	F77/F90	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FTN, .FPP, .F90

• GNU 컴파일러 사용 예제

다음은 test 예제파일을 GNU 컴파일러로 컴파일하여 실행파일 test.exe를 만드는 예시임

```
$ module load gcc/10.2.0
$ gcc -o test.exe test.c
혹은
$ gfortran -o test.exe test.f90
$ ./test.exe
```

※ /apps/shell/job_examples에서 작업제출 test 예제파일을 복사하여 사용 가능

3) PGI 컴파일러

PGI 컴파일러를 사용하기 위해서 필요한 버전의 PGI 컴파일러 모듈을 추가하여 사용한다. 사용 가능한 모듈은 module avail로 확인할 수 있다.

```
$ module load nvidia_hpc_sdk/22.7
```

※ 프로그래밍 도구 설치 현황표를 참고하여 사용 가능 버전 확인

• 컴파일러 종류

컴파일러	프로그램	소스 확장자
pgcc / pgc++	C / C++	.C, .cc, .cpp, .cxx,.c++
pgfortran	F77/F90	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FTN, .FPP, .F90

• PGI 컴파일러 사용 예제

다음은 test 예제파일을 PGI 컴파일러로 컴파일하여 실행파일 test.exe를 만드는 예시임

```
$ module load pgi/19.1
$ pgcc -o test.exe test.c
혹은
$ pgfortran -o test.exe test.f90
$ ./test.exe
```

※ /apps/shell/job_examples에서 작업제출 test 예제파일을 복사하여 사용 가능

3. 병렬 프로그램 컴파일

1) OpenMP 컴파일

OpenMP는 컴파일러 지시자만으로 멀티 스레드를 활용할 수 있도록 간단하게 개발된 기법으로 OpenMP를 사용한 병렬 프로그램 컴파일 시 사용되는 컴파일러는 순차 프로그램과 동일하며, 컴파일러 옵션을 추가하여 병렬 컴파일을 할 수 있는데, 현재 대부분의 컴파일러가 OpenMP 지시자를 지원한다.

컴파일러 옵션	프로그램	옵션
icc / icpc / ifort	C / C++ / F77/F90	-qopenmp
gcc / g++ / gfortran	C / C++ / F77/F90	-fopenmp
pgcc / pgc++ / pgfortran	C / C++ / F77/F90	-mp

• **OpenMP 프로그램 컴파일 예시 (Intel 컴파일러)**

다음은 **openMP**를 사용하는 test_omp 예제파일을 intel 컴파일러로 컴파일하여 실행파일 test_omp.exe를 만드는 예시임

```
$ module load intel/19.1.2
$ icc -o test_omp.exe -qopenmp test_omp.c
혹은
$ ifort -o test_omp.exe -qopenmp test_omp.f90
$ ./test_omp.exe
```

• **OpenMP 프로그램 컴파일 예시 (GNU 컴파일러)**

다음은 **openMP**를 사용하는 test_omp 예제파일을 GNU 컴파일러로 컴파일하여 실행파일 test_omp.exe를 만드는 예시임

```
$ module load intel/19.1.2
$ icc -o test_omp.exe -qopenmp test_omp.c
혹은
$ ifort -o test_omp.exe -qopenmp test_omp.f90
$ ./test_omp.exe
```

• **OpenMP 프로그램 컴파일 예시 (PGI 컴파일러)**

다음은 **openMP**를 사용하는 test_omp 예제파일을 PGI 컴파일러로 컴파일하여 실행파일 test_omp.exe를 만드는 예시임

```
$ module load nvidia_hpc_sdk/22.7
$ pgcc -o test_omp.exe -mp test_omp.c
혹은
$ pgfortran -o test_omp.exe -mp test_omp.f90
$ ./test_omp.exe
```

2) MPI 컴파일

사용자는 다음 표의 MPI 명령을 실행할 수 있는데, 이 명령은 일종의 wrapper로써 .bashrc를 통해 지정된 컴파일러가 소스를 컴파일하게 된다.

구분	Intel	GNU	PGI
Fortran	ifort	gfortran	pgfortran
Fortran + MPI	mpiifort	mpif90	mpif90
C	icc	gcc	pgcc
C + MPI	mpiicc	mpicc	mpicc
C++	icpc	g++	pgc++
C++ + MPI	mpiicpc	mpicxx	mpicxx

mpicc로 컴파일을 하더라도, 옵션은 wrapping되는 본래의 컴파일러에 해당하는 옵션을 사용해야 한다.

• **MPI 프로그램 컴파일 예시 (Intel 컴파일러)**

다음은 **MPI**를 사용하는 test_mpi 예제파일을 intel 컴파일러로 컴파일하여 실행파일 test_mpi.exe를 만드는 예시임

```
$ module load intel/19.1.2 mpi/impi-19.1.2
$ mpiicc -o test_mpi.exe test_mpi.c
혹은
$ mpiifort -o test_mpi.exe test_mpi.f90
$ srun ./test_mpi.exe
```

• **MPI 프로그램 컴파일 예시 (GNU 컴파일러)**

다음은 **MPI**를 사용하는 test_mpi 예제파일을 GNU 컴파일러로 컴파일하여 실행파일 test_mpi.exe를 만드는 예시임

```
$ module load gcc/10.2.0 mpi/openmpi-4.1.1
$ mpicc -o test_mpi.exe test_mpi.c
혹은
$ mpif90 -o test_mpi.exe test_mpi.f90
$ srun ./test_mpi.exe
```

• **MPI 프로그램 컴파일 예시 (PGI 컴파일러)**

다음은 **MPI**를 사용하는 test_mpi 예제파일을 PGI 컴파일러로 컴파일하여 실행파일 test_mpi.exe를 만드는 예시임

```
$ module load nvidia_hpc_sdk/22.7
$ mpicc -o test_mpi.exe test_mpi.c
혹은
$ mpifort -o test_mpi.exe test_mpi.f90
$ srun ./test_mpi.exe
```

• **CUDA + MPI 프로그램 컴파일 예시**

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1
$ mpicc -c mpi-cuda.c -o mpi-cuda.o
$ mpicc mpi-cuda.o -lcudart -L/apps/cuda/11.4/lib64
$ srun ./a.out
```

※ intel 컴파일러 사용 시, gcc/10.2.0 대신 intel/19.1.2 module을 load을 적용

Ⅰ-4. 스케줄러(SLURM)를 통한 작업 실행

Neuron 시스템의 작업 스케쥴러는 SLURM을 사용한다. 이 장에서는 SLURM을 통해 작업 제출하는 방법 및 관련 명령어들을 소개한다. SLURM에 작업 제출을 위한 작업 스크립트 작성법은 [별첨1]과 작업 스크립트 파일 작성 예시를 참고하도록 한다.

※ 사용자 작업은 /scratch/$USER에서만 제출 가능.

가. 큐 구성

- wall clock time 시간: 2일 (48시간)
- 모든 큐(파티션)에서 작업 할당은 공유 노드 정책**(하나의 노드에 복수 개 작업 동시 수행 가능)**이 적용된다.
  (22.03.17. 이후) : 자원 활용의 효율성을 위해 기존 배타적 노드 정책에서 공유 노드 정책으로 변경됨.
- 작업 큐(파티션)
  - **일반사용자가 사용할 수 있는 파티션은 jupyter, cas_v100nv_8, cas_v100nv_4, cas_v100_4, cas_v100_2, amd_a100nv_8, skl, bigmem으로 구성되어 있다. (sinfo 명령으로 노드 수, 최대 작업 실행 시간, 노드 리스트 확인 가능)**
- 작업제출 개수 제한
  - 사용자별 최대 제출 작업 개수 :
    초과하여 작업을 제출한 경우 제출 시점에 에러 발생한다.
  - 사용자별 최대 실행 작업 개수 :
    초과하여 작업을 제출한 경우 이전 작업이 끝날 때까지 기다린다.
- 리소스 점유제한
  - 작업별 최대 노드 점유 개수 :
    초과하여 작업을 제출한 경우 작업이 실행되지 않는다. 사용자의 실행 중인 여러 작업이 한 시점에 점유하고 있는 노드 개수와는 무관하다.
  - 사용자별 최대 GPU 점유 개수 :
    사용자별 총 GPU 점유 개수를 제한하는 설정으로 초과하는 경우 이전 작업이 끝날 때까지 기다린다. 사용자의 실행 중인 여러 작업이 한 시점에 점유하고 있는 GPU 개수를 제한한다.

※ 노드 구성/파티션은 시스템 사용량에 따라 시스템 운영 중에 조정될 수 있음.

나. 작업 제출 및 모니터링

1. 기본명령어 요약

명령어	내용
$ sbatch [옵션..] 스크립트	작업 제출
$ scancel 작업ID	작업 삭제
$ squeue	작업 상태 확인
$ smap	작업 상태 및 노드 상태 그래픽으로 확인
$ sinfo [옵션..]	노드 정보 확인

※ sinfo --help 명령어를 이용하여 sinfo의 옵션을 확인하실 수 있습니다.

※ Neuron 시스템 사용자 편익 증대를 위한 자료 수집의 목적으로, 아래와 같이 SBATCH 옵션을 통한 사용 프로그램 정보 작성을 의무화한다. 즉, 사용하는 어플리케이션에 맞게 SBATCH의 --comment 옵션을 아래 표를 참조하여 반드시 기입한 후 작업을 제출해야 한다.

※ 딥러닝 또는 기계학습을 위한 application을 사용하시는 경우 tensorflow, caffe, R, pytorch 등으로 구체적으로 명시해주시기 바랍니다.

※ 어플리케이션 구분을 추가는 주기적으로 수집된 사용자 요구에 맞추어 진행됩니다. 추가를 원하시면 consult@ksc.re.kr로 해당 어플리케이션에 대한 추가 요청을 해주시기 바랍니다.

[Application 별 SBATCH 옵션 이름표]

Application 종류	SBATCH 옵션 이름	Application 종류	SBATCH 옵션 이름
Charmm	charmm	LAMMPS	lammps
Gaussian	gaussian	NAMD	namd
OpenFoam	openfoam	Quantum Espresso	qe
WRF	wrf	SIESTA	siesta
in-house code	inhouse	Tensorflow	tensorflow
PYTHON	python	Caffe	caffe
R	R	Pytorch	pytorch
VASP	vasp	Sklearn	sklearn
Gromacs	gromacs	그 외 applications	etc

2. 배치 작업 제출

sbatch 명령을 이용하여 “sbatch {스크립트 파일}”과 같이 작업을 제출한다.

```
$ sbath [UserJob.script]
```

• 작업 진행 확인

할당받은 노드에 접속하여 작업 진행 여부를 확인할 수 있다.

1) squeue 명령어로 진행 중인 작업이 할당된 노드명(NODELIST)을 확인

```
$ squeue -u [userID]
JOBID PARTITION    NAME  USER   STATE    TIME   TIME_LIMI NODES NODELIST(REASON)
99792 cas_v100_4 ior   userID RUNNING 0:12 5:00:00    1       gpu25
```

2) ssh 명령을 이용하여 해당 노드에 접속

```
$ ssh gpu25
```

3) 계산노드에 진입하여 top 또는 nvidia-smi 명령어를 이용하여 작업 진행 여부 조회 가능

※ 2초 간격으로 GPU 사용률을 모니터링하는 예제

```
$ nvidia-smi -l 2
```

- 작업 스크립트 파일 작성 예시
  - SLURM에서 배치 작업을 수행하기 위해서는 SLURM 키워드들을 사용하여 작업 스크립트 파일을 작성해야 한다.

※ ‘[별첨1] 작업스크립트 파일 주요 키워드’를 참조

※ 기계학습 프레임워크 Conda 활용은 KISTI 슈퍼컴퓨팅 블로그 (http://blog.ksc.re.kr/127) 참조

• SLURM 키워드

키워드	설명
#SBATCH -J	작업명 지정
#SBATCH --time	최대 작업 수행 시간 지정
#SBATCH -o	작업 로그 파일명 지정
#SBATCH -e	에러 로그 파일명 지정
#SBATCH -p	사용할 파티션 지정
#SBATCH --comment	사용할 애플리케이션명
#SBATCH --nodelist=(노드 리스트)	작업을 수행할 노드 지정
#SBATCH --nodes=(노드 수)	작업을 수행할 노드 수 지정
#SBATCH --ntasks-per-node=(프로세스 수)	노드당 수행될 프로세스 수 지정
#SBATCH --cpus-per-task=(cpu core 수)	프로세스당 할당될 cpu core 수
#SBATCH --cpus-per-gpu=(cpu core 수)	GPU당 할당될 cpu core 수
#SBATCH --exclusive	노드를 전용으로 사용하기 위한 옵션

• 뉴론 공유 노드 정책에서 메모리 할당량 설정

뉴론 시스템 자원 활용의 효율성 및 사용자의 안정적인 작업 수행을 위하여 아래와 같이 메모리 할당량을 자동 조절

```
memory-per-node = ntasks-per-node * cpus-per-task * (단일 노드 메모리 가용량의 95% / 단일 노드 총 core 수)
```

※ '--exclusive 옵션 사용시에 단일 노드 메모리 가용량의 95%가 작업에 할당되며, 노드를 전용으로 사용할 수 있음. 단, 전용으로 사용가능한 노드가 확보a될 때까지 대기 시간이 길어질 수 있음.

• 뉴론 공유 노드 정책에서 GPU 당 CPU core 할당 개수 설정

GPU 어플리케이션의 안정적인 수행을 위해 노드당 CPU core 개수를 GPU에 비례하여 아래와 같이 기본 할당(메모리 용량도 자동으로 설정, 참조: 뉴론 공유 노드 정책에서 메모리 할당량 설정)

```
cpus-per-gpu = node의 총 core 수 / node의 총 GPU 수 * 요청 GPU 수(--gres=gpu:x)
```

※ 메모리 요구량 추가로 필요한 경우 기본 할당된 cpus-per-gpu 수보다 크게 자원을 요청하여 메모리 할당량을 확보할 수 있음.

• CPU Serial 프로그램

```
#!/bin/sh
#SBATCH -J Serial_cpu_job
#SBATCH -p skl
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

export OMP_NUM_THREADS=1

module purge
module load intel/19.1.2

srun ./test.exe

exit 0
```

※ 1노드 점유, 순차 사용 예제

• CPU OpenMP 프로그램

```
#!/bin/sh
#SBATCH -J OpenMP_cpu_job
#SBATCH -p skl
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

export OMP_NUM_THREADS=10

module purge
module load intel/19.1.2

mpirun ./test_omp.exe

exit 0
```

※ 1노드 점유, 노드당 10스레드 사용 예제

• CPU MPI 프로그램

```
#!/bin/sh
#SBATCH -J MPI_cpu_job
#SBATCH -p skl
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

module purge
module load intel/19.1.2 mpi/impi-19.1.2

mpirun ./test_mpi.exe
```

※ 2노드 점유, 노드당 4 프로세스(총 8 MPI 프로세스) 사용 예제

• CPU Hybrid (OpenMP+MPI) 프로그램

```
#!/bin/sh
#SBATCH -J hybrid_cpu_job
#SBATCH -p skl
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

module purge
module load intel/19.1.2 mpi/impi-19.1.2

export OMP_NUM_THREADS=10

mpirun ./test_mpi.exe
```

※ 1노드 점유, 노드당 2 프로세스, 프로세스당 10스레드(총 2 MPI 프로세스, 20 OpenMP 스레드) 사용 예제

• GPU Serial 프로그램

```
#!/bin/sh
#SBATCH -J Serial_gpu_job
#SBATCH -p cas_v100_4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:1 # using 2 gpus per node
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

export OMP_NUM_THREADS=1

module purge
module load intel/19.1.2 cuda/11.4

srun ./test.exe

exit 0
```

※ 1노드 점유, 순차 사용 예제

• GPU OpenMP 프로그램

```
#!/bin/sh
#SBATCH -J openmp_gpu_job
#SBATCH -p cas_v100_4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:2 # using 2 gpus per node
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

export OMP_NUM_THREADS=10

module purge
module load intel/19.1.2 cuda/11.4

srun ./test_omp.exe

exit 0
```

※ 1노드 점유, 노드당 10스레드 2GPU 사용 예제

• GPU MPI 프로그램

```
#!/bin/sh
#SBATCH -J mpi_gpu_job
#SBATCH -p cas_v100_4
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:2 # using 2 gpus per node
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

module purge
module load intel/19.1.2 cuda/11.4 cudampi/mvapich2-2.3.6

srun ./test_mpi.exe
```

※ 2노드 점유, 노드당 4 프로세스(총 8 MPI 프로세스), 노드당 2GPU 사용 예제

• GPU MPI 프로그램 - 1 node의 모든 CPU를 점유하는 실행예제

```
#!/bin/sh
#SBATCH -J mpi_gpu_job
#SBATCH -p cas_v100_4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:2
#SBATCH --comment xxx

module purge
module load intel/19.1.2 cuda/11.4 cudampi/mvapich2-2.3.6

srun ./test_mpi.exe
```

※ cas_v100_4 1개 노드 모든 core 점유, 2GPU 사용 예제

• GPU MPI 프로그램 - 1 node CPU의 절반만 점유하는 실행 예제

```
#!/bin/sh
#SBATCH -J mpi_gpu_job
#SBATCH -p cas_v100nv_8
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:4
#SBATCH --comment xxx

module purge
module load intel/19.1.2 cuda/11.4 cudampi/mvapich2-2.3.6

srun ./test_mpi.exe
```

※ cas_v100nv_8 1개 노드의 절반 core 점유, 4 GPU 사용 예제

※ 파티션별 총 core 수는 스케줄러(SLURM)를 통한 작업 실행 > 가. 큐 구성 > Total CPU core 수 참고

• 많은 메모리 할당이 필요한 프로그램 실행 예제

```
#!/bin/sh
#SBATCH -J mem_alloc_job
#SBATCH -p cas_v100_4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --time=01:00:00
#SBATCH --comment xxx #Application별 SBATCH 옵션 이름표 참고

module purge
module load intel/19.1.2 mpi/impi-19.1.2

mpirun -n 2 ./test.exe

exit 0
```

※ **프로그램 실행에 사용할 core 수는 적으나, 메모리 사용량이 큰 경우** 노드당 수행될 프로세스 수로 메모리 할당량을 조절하여 프로그램 실행하는 예제

※ '--mem' (노드 당 메모리 할당) 옵션은 사용 불가함. 노드 당 수행될 프로세스 수(ntasks-per-node)와 프로세스당 할당될 cpu core 수(cpus-per-task)를 입력하면 아래 수식에 따라 메모리 할당량이 자동 계산됨
(memory-per-node = ntasks-per-node * cpus-per-task *
(단일 노드 메모리 가용량의 95% / 단일 노드 총 core 수)

※ '--exclusive 옵션 사용 시에 단일 노드 메모리 가용량의 95%가 작업에 할당되며, 노드를 전용으로 사용할 수 있음, 단 전용으로 사용 가능한 노드가 확보될 때까지 대기 시간이 길어질 수 있음.

3. 인터렉티브 작업 제출

- 자원 할당

cas_v100_4 파티션의 gpu 2노드(각각 2core, 2gpu)를 interactive 용도로 사용

```
$ salloc --partition=cas_v100_4 --nodes=2 --ntasks-per-node=2 --gres=gpu:2 --comment={SBATCH 옵션이름}
```

※ {SBATCH 옵션이름}은 Application별 SBATCH 옵션 이름표 참고

※ 인터렉티브 작업의 walltime은 8시간으로 고정됨

- 작업 실행

```
$ srun ./(실행파일) (실행옵션)
```

- 진입한 노드에서 나가기 또는 자원 할당 취소

```
$ exit
```

- 커맨드를 통한 작업 삭제

```
$ scancel [Job_ID]
```

※ Job ID는 squeue 명령으로 확인 가능

4. 작업 모니터링

• 파티션 상태 조회

sinfo 명령을 이용하여 조회

```
$ sinfo
PARTITION     AVAIL    TIMELIMIT   NODES STATE NODELIST
jupyter         up       2-00:00:00 6        mix      jupyter[01-05,07]
cas_v100nv_8   up        2-00:00:00 4        idle     gpu[01-04]
cas_v100nv_4   up        2-00:00:00 2        alloc    gpu[06-07]
cas_v100nv_4   up        2-00:00:00 2        idle     gpu[08-09]
cas_v100_4     up        2-00:00:00 1        mix      gpu10
cas_v100_4     up        2-00:00:00 10       idle     gpu[11-20]
cas_v100_2     up        2-00:00:00 1        mix      gpu25
```

※ 노드 구성은 시스템 부하에 따라 시스템 운영 중 조정될 수 있음.

- PARTITION : 현재 SLURM에서 설정된 파티션명.
  - AVAIL : 파티션의 상태 (up or down)
  - TIMELIMIT : wall clock time
  - NODES : 노드 수
  - STATE : 노드의 상태 (alloc-자원사용중/Idle-사용가능)
  - NODELIST : 노드 리스트

• 노드별 상세 정보

sinfo 명령 뒤에 "-Nel" 옵션을 사용하면 상세 조회가 가능하다.

```
$ sinfo -Nel
Fri Mar 18 10:52:13 2022
NODELIST NODES PARTITION   STATE CPUS S:C:T  MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
gpu01    1     cas_v100nv_8 idle  32   2:16:1 384000   0         1      TeslaV10 none
gpu02    1     cas_v100nv_8 idle  32   2:16:1 384000   0         1      TeslaV10 none
gpu03    1     cas_v100nv_8 idle  32   2:16:1 384000   0         1      TeslaV10 none
gpu04    1     cas_v100nv_8 idle  32   2:16:1 384000   0         1      TeslaV10 none
- -  이 하 생 략 - -
```

• **작업 상태 조회**

squeue 명령을 이용하여 작업 목록 및 상태를 조회

```
$squeue
  JOBID PARTITION NAME     USER  ST TIME NODES. NODELIST(REASON)
  760   cas_v100_4 gpu_burn  userid R  0:00  10      gpu10
  761   cas_v100_4 gpu_burn  userid R  0:00  10      gpu11
  762   cas_v100_4 gpu_burn  userid R  0:00  10      gpu12
```

• **작업 상태 확인 및 노드 상태 그래픽으로 확인**

```
$ smap
```

• **제출된 작업 상세 조회**

```
$ scontrol show job [작업 ID]
```

다. 작업 제어

• **작업 삭제(취소)**

scancel 명령을 이용하여 “scancel [Job_ID]"과 같이 작업을 삭제한다.
Job_ID는 squeue 명령을 이용하여 조회해서 확인한다.

```
$ scancel 761
```

라. 컴파일, 디버깅, 작업제출 위치

- 로그인 노드에서 ssh로 직접 접속이 가능한 디버깅 노드를 제공하고 있음.
- 로그인/디버깅 노드에서 컴파일, 디버깅 및 모든 파티션에 대한 작업 제출이 가능함.
- 디버깅 노드는 CPU time Limit이 120분임.
- 필요시 모든 파티션에서 SLURM Interactive Job 기능을 이용해 컴파일, 디버깅이 가능함.

Ⅰ-5. 사용자 지원

사용 중 문제가 생기거나 의문 사항이 있으면 KISTI 홈페이지 > 기술지원 > **상담**을 통해 문의한다.

분야	연락처
기술지원 안내 (상담 및 최적병렬화)	https://www.ksc.re.kr/gsjw/gsjwan
교육 문의	https://kacademy.kisti.re.kr/
시스템 활용 정보	https://docs-ksc.gitbook.io/blog/

Ⅱ. 소프트웨어

Ⅱ-1. 가우시안16(Gaussian16) on GPU

본 문서는 Neuron 시스템에서 가우시안 소프트웨어 사용을 위한 기초적인 정보를 제공하고 있습니다. 따라서, 가우시안 소프트웨어 사용법 및 Neuron/리눅스 사용법 등은 포함되어 있지 않습니다. Neuron/리눅스 사용법에 대한 정보는 KISTI 홈페이지 (**https://www.ksc.re.kr**)의 기술지원 > 사용자 지침서 등을 참고하시기 바랍니다.

가. 가우시안 소개

가우시안은 에너지, 분자구조 및 진동주파수를 예측하는 분자 모델링 패키지이며, 화학, 물리, 생명과학, 공학 분야 연구자를 위한 프로그램입니다.
자세한 사항은 가우시안 사의 홈페이지를 통해 얻을 수 있습니다.
홈페이지 주소: http://gaussian.com

나. 설치 버전 및 라이선스

- KISTI 슈퍼컴퓨팅센터는 가우시안16의 사이트 라이선스를 보유하고 있으며, Neuron 시스템에는 가우시안16 Rev. A03이 설치되어 있습니다.
- 가우시안16 Rev. A03 버전은 V100 GPU를 지원하지 않으니 K40 파티션으로 작업을 제출해야 합니다. (http://gaussian.com/gpu)
- 가우시안16을 사용하기 위해서는 사용자의 계정이 가우시안 그룹(gauss group)에 등록되어야 합니다. 가우시안 그룹 등록은 KISTI 홈페이지 또는 account@ksc.re.kr로 문의하시기 바랍니다.
- 내 계정이 가우시안 그룹에 속해있는지 확인하는 방법은 다음과 같습니다.

```
$ id 사용자ID
```

※ 가우시안 그룹에 포함되어 있으면 출력 결과에 "1000009(gauss)"이 포함되어 있어야 합니다.

- 보안 문제로 사용자는 프로그램의 소스 코드에는 접근할 수 없고, 실행 파일과 기저함수(basis function)에만 접근할 수 있습니다. 실제로 프로그램을 사용하는 데는 아무런 지장이 없습니다.
- 가우시안에 연동하여 사용하는 프로그램을 사용하기 위해서는 사전에 일부 소스 코드 혹은 쉘 파일에 대한 접근권한이 필요하며 (예, Gaussrate) 이 경우 KISTI 홈페이지 또는 account@ksc.re.kr 메일을 통해 요청하셔야 합니다.
- HF 계산과 DFT 계산은 병렬로 수행할 수 있습니다.
- 가우시안16 Rev. A03 버전은 V100 GPU를 지원하지 않으며, K40 파티션으로 작업을 제출해야 하나, K40 파티션은 노드 구성 변경으로 퇴역되었으니 다른 버전을 사용하시기 바랍니다.

다. 소프트웨어 실행 방법

1. 환경설정

가우시안16은 module 명령을 통하여 환경을 로드할 수 있습니다.

```
$ module load gaussian/g16.c01
```

2. 스케쥴러 작업 스크립트 파일 작성

Neuron 시스템에서는 로그인 노드에서 SLURM이라는 스케쥴러를 사용하여 작업을 제출해야 합니다.

Neuron 시스템에서 SLURM을 사용하는 예제 파일들이 아래의 경로에 존재하므로 사용자 작업용 파일을 만들 때 이를 참고하시기 바랍니다.

- 독점 노드 방식으로 실행하는 예제 작업 스크립트 : /apps/applications/test_samples/G16/g16.c01_gpu.sh
- 공유 노드 방식으로 실행하는 예제 작업 스크립트 : /apps/applications/test_samples/G16/g16.c01_gpu_share.sh

※ 아래 예제는 Neuron 시스템 에서의 가우시안16에 대한 예제입니다.

- 파일 위치: /apps/applications/test_samples/G16/g16.c01_gpu.sh

```
#!/bin/sh
#SBATCH -J test
#SBATCH -p cas_v100_4 #cas_v100_4 노드 사용
#SBATCH --nodes=1
#SBATCH -o %x_%j.out # 표준 출력의 파일 이름을 정의
#SBATCH -e %x_%j.err # 표준 에러의 파일 이름을 정의
#SBATCH -t 00:30:00
#SBATCH --comment gaussian
#SBATCH --exclusive #노드를 독점으로 할당받아 사용
#SBATCH --gres=gpu:4

module purge
module load gaussian/g16.c01

export GAUSS_SCRDIR="$SLURM_SUBMIT_DIR"

export GAUSS_MDEF=96GB

export GAUSS_CDEF="0-39"
export GAUSS_GDEF="0-3=0,10,11,19"

g16 test.com
```

- GAUSS_CDEF 변수는 %CPU 옵션과 동일하며, 입력 파일에 %CPU 값이 있을 경우 해당 값이 적용됩니다.
- 과거 배타적 노드 정책에서는 한 노드당 하나의 작업만이 독점적으로 할당되었기 때문에 특정 CPU를 지정하여 사용이 가능하였으나, 공유 노드 정책에서는 한 노드에 여러 작업이 수행될 수 있으며 작업에서 할당받은 CPU의 적절한 지정이 필요합니다. 위 예제에서는 cas_v100_4 노드를 독점적으로 할당받아 40 core와 4 GPU를 할당받아 작업 수행하는 예시입니다.
- 공유 노드에서의 작업 스크립트 예시는 /apps/applications/test_samples/G16/g16.c01_gpu_share.sh 파일 참고 바랍니다.
- GAUSS_GDEF 변수는 %GPUCPU 옵션과 동일하며, GPU와 CPU의 맵핑 옵션으로 0번 GPU(첫번째 GPU, nvidia-smi 명령으로 확인)의 제어 CPU를 0번 CPU로 지정함을 의미합니다. (%GPUCPU=gpu-list=control-cpus)
- GAUSS_MDEF 변수는 %Mem 옵션과 동일하며, 입력 파일에 %Mem 값이 있을 경우 해당 값이 적용됩니다. 계산 규모에 따라 적절한 값을 입력해주시기 바라며, slurm 스케줄러에서 할당받은 메모리를 초과하지 않는 범위 내에서 설정하시기 바랍니다. (뉴론 사용자 지침서 > 스케줄러(SLURM)를 통한 작업 실행)
- 참고 : http://gaussian.com/relnotes

※ 가우시안 입력 파일을 PC에서 작성 후 FTP로 전송한다면, 반드시 ascii mode로 전송해야만 합니다.

- 기타 SLURM에 관련된 명령어 및 사용법은 Neuron 사용자 지침서를 참조하시면 됩니다.

- SLURM 스케쥴러의 주요 명령어 (세부 사항은 Neuron 시스템 사용자 지침서 참조)

명령어	설명
sbatch g16_gpu.sh	SLURM에 작업스크립트(g16_gpu.sh)를 제출
squeue	SLURM에 제출되어 대기/수행 중인 전체 작업 조회
sinfo	SLURM 파티션 정보 조회
scancel job-id	작업 제출한 작업 취소

라. 참고자료

- 가우시안을 처음으로 사용하고자 하는 사람은 다음의 책의 일독을 권합니다.
  - James B. Foresman and Aeleen Frisch,
    "Exploring Chemistry with Electronic Structure Methods:
    A Guide to Using Gaussian",
    www.amazon.com, www.bn.com 등의 온라인 서점에서 구매할 수 있고,
    http://gaussian.com에서도 직접 구매가 가능합니다.
- 가우시안에 관한 모든 정보는 Gaussian사의 홈페이지(http://gaussian.com)를 통해 얻을 수 있습니다.

Ⅲ. 부록

Ⅲ-1. 작업 스크립트 주요 키워드

작업 스크립트 내에서 적절한 키워드를 사용하여 원하는 작업을 위한 자원 할당 방법을 명시해야 한다. 주요 키워드는 아래와 같으며, 사용자는 이들 중에서 몇 가지만 사용하여 작업 스크립트 파일을 작성할 수 있다.

• **job-name (-J, --job-name)**
작업의 이름을 지정하며, 명시하지 않으면 스크립트 파일 이름이 작업 이름으로 지정된다.

• **time (-t, --time)**
예상되는 작업 소요 시간을 의미하며, 실제 예상되는 작업 소요 시간보다 약간 더 길게 설정해 주는 것이 안전하다. 해당 파티션의 Wall time limit을 초과하면, 작업이 제출되지 않는다. 지정된 시간에 이르렀는데 작업이 완료되지 않으면 SLURM 스케줄러가 작업을 강제 종료시킨다.

• **partition (-p, --partition)**
작업 수행을 위한 SLURM 파티션을 지정한다. 파티션명은 sinfo 명령어로 확인이 가능하다.

• **nodes (-N, --nodes)**
작업을 위해 할당할 노드의 수를 지정한다.

• **ntasks (-n, --ntasks)**
작업을 위해 할당할 프로세스의 수를 지정한다.

• **ntasks-per-node (--ntasks-per-node)**
노드당 할당할 프로세스의 수를 지정한다.

• **input (-i, --input)**
Standard input을 지정한다.

• **cpus-per-task (-c, --cpus-per-task)**
작업 태스크 당 필요한 CPU 개수를 명시한다.

• **output (-o, --output)**

Standard output을 지정한다.

- %x : "job name"으로 지정한 명칭을 파일명으로 사용한다.
- %j : 작업 제출 시 부여되는 "job ID"를 파일명으로 사용한다.
- %a : "job array ID" (index) 번호를 파일명으로 사용한다.
- %u : "user ID"를 파일명으로 사용한다.

• **error (-e, --error)**

Standard error를 지정한다.

- %x : "job name"으로 지정한 명칭을 파일명으로 사용한다.
- %j : 작업 제출 시 부여되는 "job ID"를 파일명으로 사용한다.
- %a : "job array ID" (index) 번호를 파일명으로 사용한다.
- %u : "user ID"를 파일명으로 사용한다.

• **dependency (-d, --dependency)**

작업 의존성을 설정한다. 설정된 작업이 종료된 후에 작업이 시작된다.

※ 상세 매뉴얼 : http://slurm.schedmd.com/ 참조

Ⅲ-2. Conda

아나콘다(Anaconda)는 PYTHON과 R 프로그래밍 언어로 된 과학 컴퓨팅(데이터 과학, 기계 학습 응용 프로그램, 대규모 데이터 처리, 예측 분석 등) 분야의 패키지들의 모음을 제공하는 배포판이다.

Anaconda 배포판은 1,200 만 명이 넘는 사용자가 사용하며 Windows, Linux 및 MacOS에 적합한 1400가지 이상의 인기 있는 데이터 과학 패키지를 포함한다.

Anaconda를 설치하기 위해서는 https://www.anaconda.com 웹 사이트에서 자신의 OS에 맞는 배포판을 다운받아 설치하면 된다.

(예) Windows, MacOS, Linux

현재 Anaconda는 Python 3.7 기반의 버전과 Python 2.7 기반의 버전을 제공한다.

conda는 아나콘다에서 패키지 버전 관리를 위해 제공되는 어플리케이션이다.

Python 사용자들이 패키지 설치 시 가장 어려움을 겪는 의존성 문제를 conda를 활용함으로써 쉽게 해결할 수 있다.

본 문서는 KISTI 시스템에서 Python 사용자를 위하여 conda 패키지 활용하는 방법을 소개한다.

소개 페이지의 "/home01/userID" 는 테스트 계정의 홈 디렉토리로 자신에 맞는 경로로 적절히 변경해서 사용해야 한다.

가. Conda의 사용

- Miniconda는 https://docs.conda.io/en/latest/miniconda.html 사이트에서 각 OS에 맞는 버전을 다운받을 수 있고,
- Anaconda는 https://www.anaconda.com/distribution/#download-section 사이트에서 각 OS에 맞는 버전을 다운받을 수 있다.

명령어 모음	내용
clean	Remove unused packages and caches.
config	Modify configuration values in .condarc. This is modeled after the git config command. Writes to the user .condarc file (/home01/userID/.condarc) by default.
create	Create a new conda environment from a list of specified packages.
help	Displays a list of available conda commands and their help strings.
info	Display information about current conda install.
init	Initialize conda for shell interaction. [Experimental]
install	Installs a list of packages into a specified conda environment.
list	List linked packages in a conda environment.
package	Low-level conda package utility. (EXPERIMENTAL)
remove	Remove a list of packages from a specified conda environment.
uninstall	Alias for conda remove.
run	Run an executable in a conda environment. [Experimental]
search	Search for packages and display associated information. The input is a MatchSpec, a query language for conda packages. See examples below.
update	Updates conda packages to the latest compatible version.
upgrade	Alias for conda update

- Conda Initialize 방법

conda init 명령으로 홈 디렉터리의 .bashrc에 설정을 추가할 수 있다.

```
$ source /apps/applications/Miniconda/23.3.1/etc/profile.d/conda.sh
$ conda init
$ source ~/.bashrc
```

- conda 경로 변경 방법

conda 환경, 패키지 경로는 기본적으로 홈 디렉터리로 설정되어 있으나, scratch와 같은 다른 경로로도 변경할 수 있다.

```
$ echo "export CONDA_ENVS_PATH=/scratch/$USER/.conda/envs" >> /home01/$USER/.bashrc
$ echo "export CONDA_PKGS_DIRS=/scratch/$USER/.conda/pkgs" >> /home01/$USER/.bashrc
$ source ~/.bashrc
```

나. Conda Environment 생성

- conda environment는 Python의 독립적인 가상 실행 환경을 만들어 패키지들의 버전 관리에 용이하다.
- "conda create -n [ENVIRONMENT]" 을 이용하여 conda environment를 생성할 수 있다.
- 기본값으로 conda path의 envs 아래 경로에 지정한 environment 이름으로 생성된다.
- "--use-local" 옵션을 사용하면
  사용자 홈 디렉토리(**${HOME}/.conda/envs/[environment_name]**)에 생성된다.

```
※ conda initialize를 하지 않았다면 최초 1회 실행
$ source /apps/applications/Miniconda/23.3.1/etc/profile.d/conda.sh
$ conda init
$ source ~/.bashrc

$ conda create -n scikit-learn_0.21
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home01/userID/.conda/envs/scikit-learn_0.21

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use:
# > conda activate scikit-learn_0.21
#
# To deactivate an active environment, use:
# > conda deactivate
#

$ conda activate scikit-learn_0.21
(scikit-learn_0.21) $
```

다. Conda Environment에 패키지 설치 및 확인

- conda install [패키지명]으로 패키지를 설치할 수 있다.
- conda 채널에 있는 패키지는 "conda install -c [채널명] [패키지명]"와 같이 설치할 수 있다.
- 위 "2" 항목에서 생성한 conda environment 경로 아래에 패키지들이 설치된다.
- 예제

```
$ source activate scikit-learn_0.21
(scikit-learn_0.21) $ conda install scikit-learn
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home01/userID/.conda/envs/scikit-learn_0.21
  added / updated specs:
    - scikit-learn

The following packages will be downloaded:
    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2019.1.23  |                0         126 KB
    …
      wheel-0.33.1               |           py37_0          39 KB
    ------------------------------------------------------------
                                           Total:        277.6 MB

The following NEW packages will be INSTALLED:

  blas                pkgs/main/linux-64::blas-1.0-mkl
 …
   zlib                pkgs/main/linux-64::zlib-1.2.11-h7b6447c_3

Proceed ([y]/n)? y

Downloading and Extracting Packages
setuptools-40.8.0    | 643 KB    | ##################################### | 100%
…
```

```
openssl-1.1.1b        | 4.0 MB    | ######################################## | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(scikit-learn_0.21) $ python -c "import sklearn"
(scikit-learn_0.21) $
```

라. Conda Environment 목록 확인

• "conda-env list" 또는 "conda env list"를 이용하여 목록을 확인할 수 있다.

```
(scikit-learn_0.21) $ conda env list
# conda environments:
#
base                     /apps/applications/PYTHON/3.7.1
scikit-learn_0.21      *  /home01/userID/.conda/envs/scikit-learn_0.21
(scikit-learn_0.21) $ conda deactivate
$
```

마. Conda Environment 내보내기

• 내보내기 전 conda-pack 패키지 필요

※ (참고) https://conda.github.io/conda-pack

• "conda pack -n [ENVIRONMENT] -o [파일명]" 을 이용하여 conda environment를 다른 시스템에서 활용할 수 있다.

• (예) 외부 인터넷이 연결되지 않는 경우, 다른 시스템에서 동일한 conda 환경을 이용하는 경우

```
$ conda activate scikit-learn_0.21
(scikit-learn_0.21) $ conda install -c conda-forge conda-pack
(scikit-learn_0.21) $ conda pack -n scikit-learn_0.21 -o scikit-learn_test.tar.gz
Collecting packages...
Packing environment at '/home01/userID/.conda/envs/scikit-learn_0.21' to 'scikit-learn_test.tar.gz'
[########################################] | 100% Completed |  4min 18.8s
(scikit-learn_0.21) $ ls -l scikit-learn_test.tar.gz
-rw-------. 1 userID in0162 1459826406 Mar 28 15:03 scikit-learn_test.tar.gz
(scikit-learn_0.21) $
```

바. Conda Environment 가져오기

- conda pack을 이용하여 생성했던 conda environment를 아래 -예제-와 같이 가져와 환경설정 후 사용 가능.

```
$ mkdir -p $HOME/.conda/envs/scikit-learn_test
$ tar xvzf scikit-learn_test.tar.gz -C $HOME/.conda/envs/scikit-learn_test
※ scratch 가 conda 경로인 경우, /scratch/$USER/.conda 로 입력

$ conda activate scikit-learn_test
(scikit-learn_0.21) $ conda-unpack
(scikit-learn_0.21) $ conda deactivate
$
```

사. Conda Environment 삭제

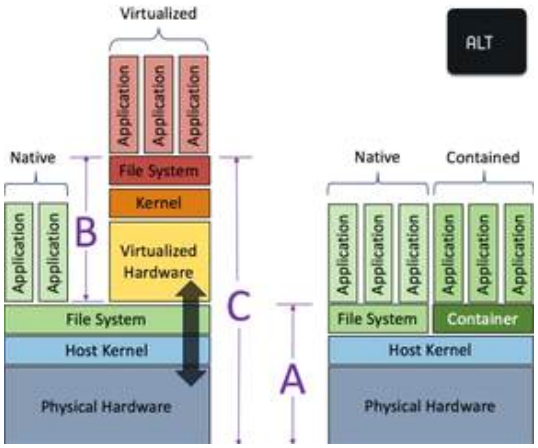
- "conda-env remove -n [ENVIRONMENT]" 또는 "conda env remove -n [ENVIRONMENT]"를 이용하여 삭제할 수 있다.

```
$ conda env remove -n scikit-learn_test
Remove all packages in environment /home01/userID/.conda/envs/scikit-learn_test:
```

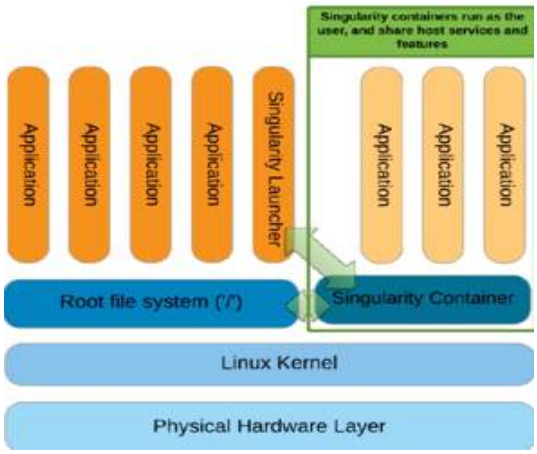
Ⅲ-3. Singularity 컨테이너

싱귤레러티(Singularity)는 도커(Docker)와 같이 OS 가상화를 구현하기 위한 HPC 환경에 적합한 컨테이너 플랫폼이다. 사용자 작업 환경에 적합한 리눅스 배포판, 컴파일러, 라이브러리, 애플리케이션 등을 포함하는 컨테이너 이미지를 빌드하고 빌드된 컨테이너 이미지를 구동하여 사용자 프로그램을 실행할 수 있다.

Tensorflow, Caffe, Pytorch와 같은 딥러닝 프레임워크와 Quantum Espresso, Lammps, Gromacs, Paraview 등을 지원하는 빌드된 **컨테이너 이미지는 /apps/applications/singularity_images/ngc 디렉터리**에서 액세스할 수 있다.



[가상머신과 컨테이너 아키텍처 비교]



[Singularity 컨테이너 아키텍처]

※ 가상머신은 애플리케이션이 하이퍼바이저와 게스트 OS를 거쳐 올라가는 구조이나, 컨테이너는 물리적인 하드웨어에 더 가까우며 별도의 게스트 OS가 아닌 호스트 OS를 공유하기 때문에 오버헤드가 더 작음. 최근 클라우드 서비스에서 컨테이너의 활용이 증가하고 있음.

가. 컨테이너 이미지 빌드하기

1. 싱귤레러티 모듈 적재 혹은 경로 설정

```
$ module load singularity/3.11.0
or
$ $HOME/.bash_profile
export PATH=$PATH:/apps/applications/singularity/3.11.0/bin/
```

2. 로컬 빌드

- 뉴론 시스템의 로그인 노드에서 컨테이너 이미지를 로컬 빌드하기 위해서는, 먼저 **KISTI 홈페이지 > 기술지원 > 상담신청**을 통해 아래와 같은 내용으로 fakeroot 사용 신청을 해야 함.
  - 시스템명 : 뉴론
  - 사용자 ID : a000abc
  - 요청 사항 : 싱귤레러티 fakeroot 사용 설정
- **NGC(Nvidia GPU Cloud)에서 배포하는 도커 컨테이너**로부터 뉴론 시스템의 Nvidia GPU에 최적화된 딥러닝 프레임워크 및 HPC 애플리케이션 관련 싱규레러티 컨테이너 이미지를 빌드할 수 있음
- 생성된 싱귤레러티 이미지 파일(*.sif)을 수정하기 위해서는 root 권한이 필요하며, 샌드박스(쓰기 가능 chroot 디렉터리)로 변환해야 함.

[이미지 빌드 명령어]

```
$ singularity [global options...] build [local options...] <IMAGE PATH> <BUILD SPEC>

[주요 global options]
    -d : 디버깅 정보를 출력함
    -v : 추가 정보를 출력함
    --version : 싱귤레러티 버전 정보를 출력함

[관련 주요 local options]
    --fakeroot : roor 권한 없이 일반사용자가 가짜 root 사용자로 이미지 빌드
    --remote : 외부 싱귤레러티 클라우드(Sylabs Cloud)를 통한 원격 빌드(root 권한 필요 없음)
    --sandbox : 샌드박스 형태의 쓰기 가능한 이미지 디렉터리 빌드

<IMAGE PATH>
   default : 읽기만 가능한 기본 이미지 파일(예시 : ubuntu1.sif)
   sandbox : 읽기 및 쓰기 가능한 디렉터리 구조의 컨테이너(예시 : ubuntu4)

<BUILD SPEC>
definition file : 컨테이너를 빌드하기 위해 recipe를 정의한  파일(예시 : ubuntu.def)
local image : 싱귤레러티 이미지 파일 혹은 샌드박스 디렉터리(IMAGE PATH 참조)
URI
library:// 컨테이너 라이브러리 (default https://cloud.sylabs.io/library)
docker:// 도커 레지스트리 (default 도커 허브)
shub:// 싱규레러티 레지스트리 (default 싱귤레러티 허브)
oras:// OCI 레지스트리
```

[예제]

```
① Definition 파일로부터 ubuntu1.sif 이미지 빌드하기
 $ singularity build --fakeroot ubuntu1.sif ubuntu.def*

② 싱귤레러티 라이브러리로부터 ubuntu2.sif 이미지 빌드하기
 $ singularity build --fakeroot ubuntu2.sif library://ubuntu:18.04

③ 도커 허브로부터 ubuntu3.sif 이미지 빌드하기
 $ singularity build --fakeroot ubuntu3.sif docker://ubuntu:18.04

④ 도커 허브로부터 샌드박스 형태의 ubuntu4 이미지 디렉터리 빌드하기
 $ singularity build --fakeroot --sandbox ubuntu4 docker://ubuntu:18.04

⑤ NGC(Nvidia GPU Cloud) 도커 레지스트리로부터 '22년 03월 배포 pytorch 이미지 빌드하기
 $ singularity build --fakeroot pytorch1.sif docker://nvcr.io/nvidia/pytorch:22.03-py3

⑥ Definition 파일로부터 pytorch.sif 이미지 빌드하기
 $ singularity build --fakeroot pytorch2.sif pytorch.def**

⑦ fakeroot 사용하지 않고 Definition 파일로부터 pytorch.sif 이미지 빌드하기
    # singularity 3.11.0 버전 이상에서 지원
    # Definition 파일에서 기존 컨테이너 이미지 파일을 기반으로 패키지 설치에 적합
 $ singularity build pytorch2.sif pytorch.def**

* ) ubuntu.def 예시
 bootstrap: docker
 from: ubuntu:18.04
 %post
 apt-get update
 apt-get install -y wget git bash gcc gfortran g++ make file
 %runscript
 echo "hello world from ubuntu container!"

** ) pytorch.def 예시
 # 로컬 이미지 파일로부터 콘다를 사용하여 새로운 패키지 설치를 포함한 이미지 빌드
 bootstrap: localimage
 from: /apps/applications/singularity_images/ngc/pytorch:22.03-py3.sif
```

```
%post
conda install matplotlib -y

# 외부 NGC 도커 이미지로부터 콘다를 사용하여 새로운 패키지 설치를 포함한 이미지 빌드
bootstrap: docker
from: nvcr.io/nvidia/pytorch:22.03-py3
%post
conda install matplotlib -y
```

3. 원격빌드

① Sylabs Cloud에서 제공하는 원격 빌드 서비스를 이용하여 Definition 파일로부터 ubuntu4.sif 이미지 빌드하기

```
$ singularity build --remote ubuntu4.sif ubuntu.def
```

※ Sylabs Cloud(https://cloud.sylabs.io)에서 제공하는 원격빌드 서비스를 이용하려면 액세스 토큰을 생성하여 뉴론 시스템에 등록해야 함. **[참조 1]**

※ 또한, Sylabs Cloud에 웹 브라우저 접속을 통해서 싱귤레러티 컨테이너 이미지의 생성·관리가 가능함. **[참조 2]**

4. 컨테이너 이미지 가져오기/내보내기

```
① Sylabs cloud 라이브러리에서 컨테이너 이미지 가져오기
$ singularity pull tensorflow.sif library://dxtr/default/hpc-tensorflow:0.1

② 도커 허브에서 이미지를 가져와서 싱귤레러티 이미지로 변환
$ singularity pull tensorflow.sif docker://tensorflow/tensorflow:latest

③ Sylabs Cloud 라이브러리에 싱귤레러티 이미지 내보내기(업로드)
$ singularity push -U tensorflow.sif library://ID/default/tensorflow.sif
```

※ Sylabs Cloud 라이브러리에 컨테이너 이미지를 내보내기(업로드) 위해서는 먼저 액세스 토큰을 생성하여 뉴론 시스템에 등록해야 함. **[참조 1]**

5. 컨테이너 이미지에서 제공되지 않는 파이썬 패키지 등을 사용자 홈 디렉터리에 설치하는 방법

① pip install --user [파이썬 패키지 이름/버전], 사용자의 /home01/ID/.local 디렉터리에 설치됨

```
$ module load ngc/tensorflow:20.09-tf1-py3 (텐서플로우 컨테이너 모듈 로드)
$ pip install --user keras==2.1.2
$ pip list --user
Package Version
------------ -------
Keras 2.1.2
```

② conda install –use-local [콘다 패키지 이름/버전], 사용자의 /home01/ID/.conda/pkgs 디렉터리에 설치됨

```
$ module load ngc/pytorch:20.09-py3 (파이토치 컨테이너 모듈 로드)
$ conda install --use-local matplotlib -y
$ conda list matplotlib
# Name Version Build Channel
matplotlib 3.3.3 pypi_0 pypi
```

※ 단, 여러 가지 컨테이너 이미지를 사용하는 경우 사용자 프로그램 실행 시 사용자 홈 디렉터리에 추가로 설치한 패키지를 먼저 찾기 때문에 다른 컨테이너 이미지에서 요구하는 패키지와 충돌이 발생하여 정상적으로 동작하지 않을 수 있음.

나. 싱귤레러티 컨테이너에서 사용자 프로그램 실행

1. 싱귤레러티 모듈 적재 혹은 경로 설정

```
$ module load singularity/3.11.0
or
$ $HOME/.bash_profile
export PATH=$PATH:/apps/applications/singularity/3.11.0/bin/
```

2. 싱귤레러티 컨테이너에서 프로그램 실행 명령어

```
$ singularity [global options...] shell [shell options...] <container>
$ singularity [global options...] exec [exec options...] <container> <command>
$ singularity [global options...] run [run options...] <container>
```

[예제]

① Nvidia GPU 장착 계산 노드의 싱귤레러티 컨테이너에서 쉘 실행 후 사용자 프로그램 실행

```
$ singularity shell --nv * tensorflow_22.03-tf1-keras-py3.sif
Singularity> python test.py
```

② Nvidia GPU 장착 계산 노드의 싱귤레러티 컨테이너에서 사용자 프로그램 실행

```
$ singularity exec --nv tensorflow_22.03-tf1-keras-py3.sif python test.py
$ singularity exec --nv docker://tensorflow/tensorflow:latest python test.py
$ singularity exec --nv library://dxtr/default/hpc-tensorflow:0.1 python test.py
```

③ Nvidia GPU 장착 계산 노드의 싱귤레러티 컨테이너에서 runscript(이미지 빌드 시 생성)가 존재하면 이 스크립트를 먼저 실행한 후, 사용자 명령어(아래 예제에서 python --version)가 존재하면 이어서 실행됨

```
$ singularity run --nv /apps/applications/singularity_images/ngc/tensorflow_22.03-tf1-keras-py3.sif \
 python --version
================
== TensorFlow ==
================

NVIDIA Release 22.03-tf1 (build 33659237)
TensorFlow Version 1.15.5

Container image Copyright (c) 2022, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
Copyright 2017-2022 The TensorFlow Authors. All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

NOTE: CUDA Forward Compatibility mode ENABLED.
 Using CUDA 11.6 driver version 510.47.03 with kernel driver version 460.32.03.
 See https://docs.nvidia.com/deploy/cuda-compatibility/ for details.

NOTE: Mellanox network driver detected, but NVIDIA peer memory driver not
 detected. Multi-node communication performance may be reduced.

Python 3.8.10 (default, Nov 26 2021, 20:14:08)
```

※ 싱귤레러티의 명령어[shell | exec | run | pull ...]별 도움말을 보려면 “singularity help [command]”를 실행함.

※ 계산/로그인 노드에서 Nvidia GPU를 사용하기 위해서는 --nv 옵션을 사용해야 함.

3. NGC 컨테이너 모듈을 사용하여 사용자 프로그램 실행

모듈 명령어를 사용하여 NGC 싱규레러티 컨테이너 이미지와 관련된 모듈을 로드하면 싱귤레러티 명령어를 입력하지 않아도 자동으로 컨테이너 이미지가 구동되어 좀 더 쉽게 싱귤레러티 컨테이너에서 사용자 프로그램을 실행할 수 있음.

• NGC 컨테이너 모듈을 로드하여 컨테이너에서 사용자 프로그램 실행하기

```
# ① tensorflow 1.15.5 지원 싱규레러티 컨테이너 이미지(tensorflow_22.03-tf1-keras-py3.sif)를
#     자동으로 구동하여 사용자 프로그램 실행
 $ module load singularity/3.9.7 ngc/tensorflow:22.03-tf1-py3
 $ mpirun -H gpu39:2,gpu44:2 -n 4 python keras_imagenet_resnet50.py

# ② lammps 지원 싱규레러티 컨테이너 이미지(lammps:15Jun2020-x86_64.sif)를
#    자동으로 구동하여 lammps 실행
 $ module load singularity/3.6.4 ngc/lammps:15Jun2020
 $ mpirun -H gpu39:2,gpu44:2 -n 4 lmp -in in.lj.txt -var x 8 -var y 8 -var z 8 -k on g 2 \
 -sf kk -pk kokkos cuda/aware on neigh full comm device binsize 2.8

# ③ gromacs 지원 싱규레러티 컨테이너 이미지(gromacs:2020.2-x86_64.sif)를 자동으로
#    구동하여 gromacs 실행
 $ module load singularity/3.6.4 ngc/gromacs:2020.2
 $ gmx mdrun -ntmpi 2 -nb gpu -ntomp 1 -pin on -v -noconfout -nsteps 5000 \
 -s topol.tpr singularity shell --nv *  tensorflow:20.09-tf1-py3.sif
```

※ 컨네이너 이미지 모듈 로드 후 실행명령어 입력만으로 “singularity run --nv <컨테이너> [실행명령어]”가 자동 실행됨.

• NGC 컨테이너 모듈 리스트

※ NGC(https://ngc.nvidia.com)에서 Nvidia GPU에 최적화하여 빌드 배포한 도커 컨테이너 이미지를 싱귤레러티로 변환함.

※ 컨테이너 이미지 파일 경로 : /apps/applications/singularity_images/ngc

```
1009% [a123a01@glogin01 ngc]$ module av
```

-- [중략] --

```
---------------------------------------------------- /apps/Modules/modulefiles/test -----------------------------------------
  amd/mvapich2_amd-2.3           cuda/10.2                      intel/oneapi_21.4        ngc/tensorflow:22.03-tf1-py3
  amd/openmpi_amd-3.1.5    (D)   cuda/11.2                (D)   mpi/oneapi_21.4          ngc/tensorflow:22.03-tf2-py3 (D)
  cistem/1.0.0-bata              cudampi/mvapich2-gdr-2.3.4     ngc/caffe:20.03-py3      python/3.9.5                 (D)
  conda/gdr_pytorch_1.4.0        cudampi/mvapich2-2.3.4         ngc/gromacs:2020.2       qe/6.4
  conda/gdr_tensorflow_1.13      cudampi/openmpi-4.0.5    (D)   ngc/lammps:29Oct2020     relion/3.0.7
  cuda/9.0                       gcc/7.2.0                      ngc/paraview:5.9.0-py3
  cuda/9.1                       gcc/9.2.0                (D)   ngc/pytorch:22.03-py3
  cuda/10.1                      git/2.9.3                      ngc/qe:6.7
```

4. 스케줄러(SLURM)를 통한 컨테이너 실행 방법

• GPU 싱귤레러티 컨테이너 작업 실행

1) 작업 스크립트를 작성하여 배치 형태 작업 실행

• 실행명령어 : sbatch < 작업 스크립트 파일 >

```
[id@glogin01]$ sbatch job_script.sh
Submitted batch job 12345
```

※ 자세한 스케줄러(SLURM) 사용 방법은 "뉴론 지침서-스케줄러(SLURM)를 통한 작업실행" 참조.

※ **[참조 3]**을 통해 병렬 학습 실행 예제 프로그램을 따라해 볼 수 있음.

2) 스케줄러가 할당한 계산 노드에서 인터랙티브 작업 실행

• 스케줄러를 통해 계산노드를 할당받아 첫 번째 계산노드에 쉘 접속 후 인터렉티브 모드로 사용자 프로그램 실행

```
[id@glogin01]$ srun --partition=cas_v100_4 –nodes=1 –ntasks-per-node=2 \
 –cpus-per-task=10 --gres=gpu:2 --comment=pytorch --pty bash
[id@gpu10]$
[id@gpu10]$ module load singularity/3.11.0
[id@gpu10]$ singularity run --nv /apps/applications/singularity_images/ngc/pytorch_22.03-hd-py3.sif \
python test.py
```

※ 1노드 점유, 노드당 2 타스크, 타스크당 10 CPUs, 노드당 2GPU 사용 예제

• GPU 싱귤레러티 컨테이너 작업 스크립트 예시

1) 단일 노드

• 실행명령어 : singularity run --nv <컨테이너> [사용자 프로그램 실행명령어]

```
#!/bin/sh
#SBATCH -J pytorch # job name
#SBATCH --time=1:00:00 # wall_time
#SBATCH -p cas_v100_4
#SBATCH --comment pytorch # application name
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH —gres=gpu:2 # number of GPUs per node

module load singularity/3.11.0

singularity run --nv /apps/applications/singularity_images/ngc/pytorch_22.03-hd-py3.sif \
python test.py
```

※ 1노드 점유, 노드당 2 타스크, 타스크당 10 CPUs, 노드당 2GPU 사용 예제

2) 멀티 노드-1

• 실행명령어 : srun singularity run --nv <컨테이너> [사용자 프로그램 실행명령어]

```
#!/bin/sh
#SBATCH -J pytorch_horovod # job name
#SBATCH --time=1:00:00 # wall_time
#SBATCH -p cas_v100_4 # partition name
#SBATCH --comment pytorch # application name
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

module load singularity/3.11.0 gcc/4.8.5 mpi/openmpi-3.1.5

srun singularity run --nv /apps/applications/singularity_images/ngc/pytorch_22.03-hd-py3.sif \
python pytorch_imagenet_resnet50.py
```

※ 2노드 점유, 노드당 2 타스크(총 4개 MPI 프로세스-horovod 사용), 타스크당 10CPUs, 노드당 2GPU 사용 예제

3) 멀티 노드-2

- NGC 컨테이너 모듈을 로드하면 사용자 프로그램 실행 시 지정된 싱귤레러티 컨테이너를 자동으로 구동함
- 실행명령어 : mpirun_wrapper [사용자 프로그램 실행명령어]

```
#!/bin/sh
#SBATCH -J pytorch_horovod # job name
#SBATCH --time=1:00:00 # wall_time
#SBATCH -p cas_v100_4 # partition name
#SBATCH --comment pytorch # application name
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

module load singularity/3.11.0  ngc/pytorch:22.03-py3

mpirun_wrapper python pytorch_imagenet_resnet50.py
```

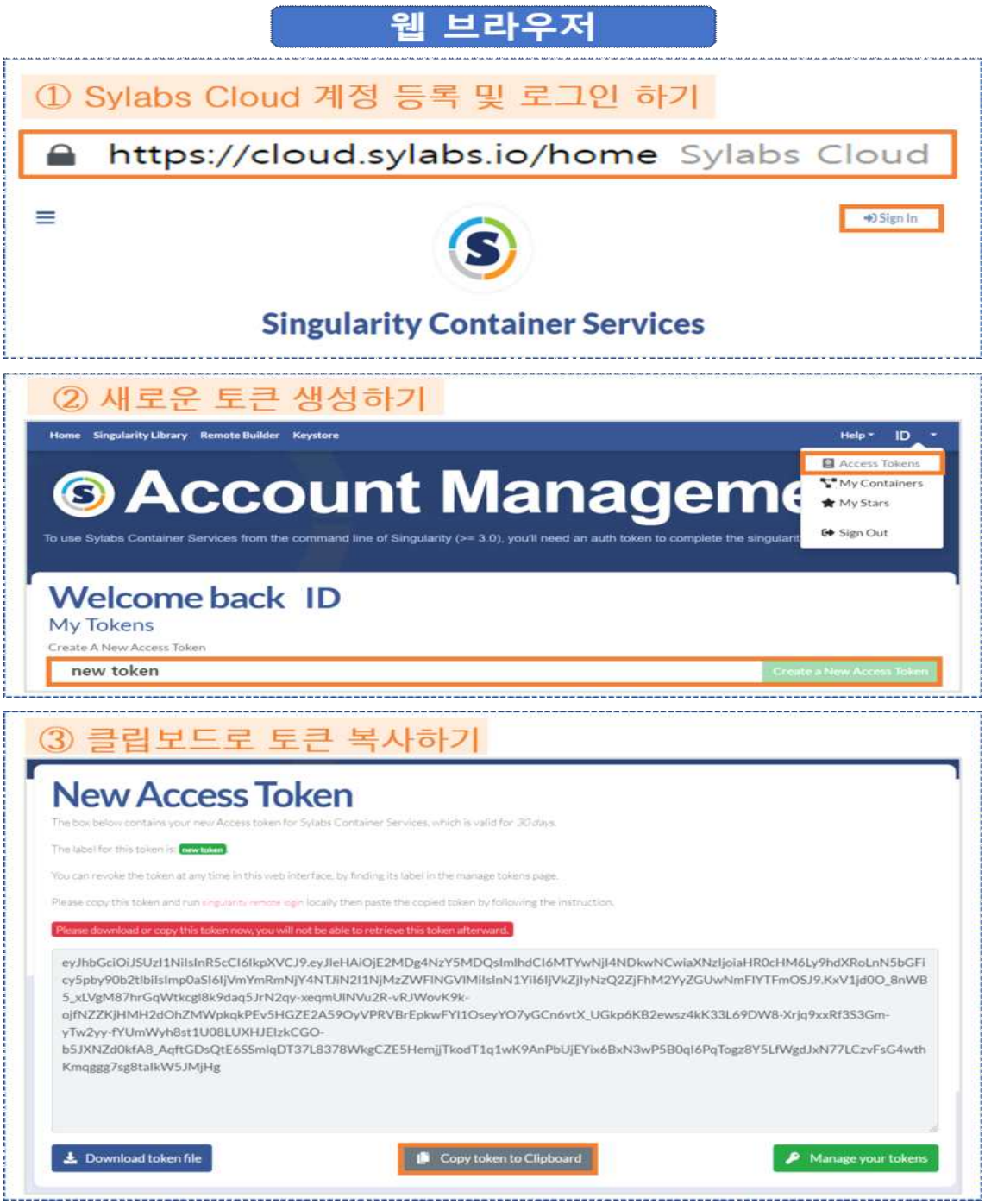
※ 2노드 점유, 노드당 2 타스크(총 4개 MPI 프로세스-horovod 사용), 타스크당 10CPUs, 노드당 2GPU 사용 예제

다. 참조

[참조1]

Sylabs Cloud 액세스 토큰 생성 및 뉴론 시스템에 토큰 등록하기

[Sylabs Cloud 바로가기]



누리온 로그인 노드

④ 토큰 입력하기

방법 1) singularity 명령어로 토큰 입력하기

```
[ID@login01 ~]$ singularity remote login
INFO:    Authenticating with default remote.
Generate an API Key at https://cloud.sylabs.io/auth/tokens, and paste here:
API Key:
INFO:    API Key Verified!
```

방법 2) 홈 디렉터리의 토큰 파일에 저장하기

```
[ID@login01 ~]$ vi /home01/ID/.singularity/sylabs-token
```

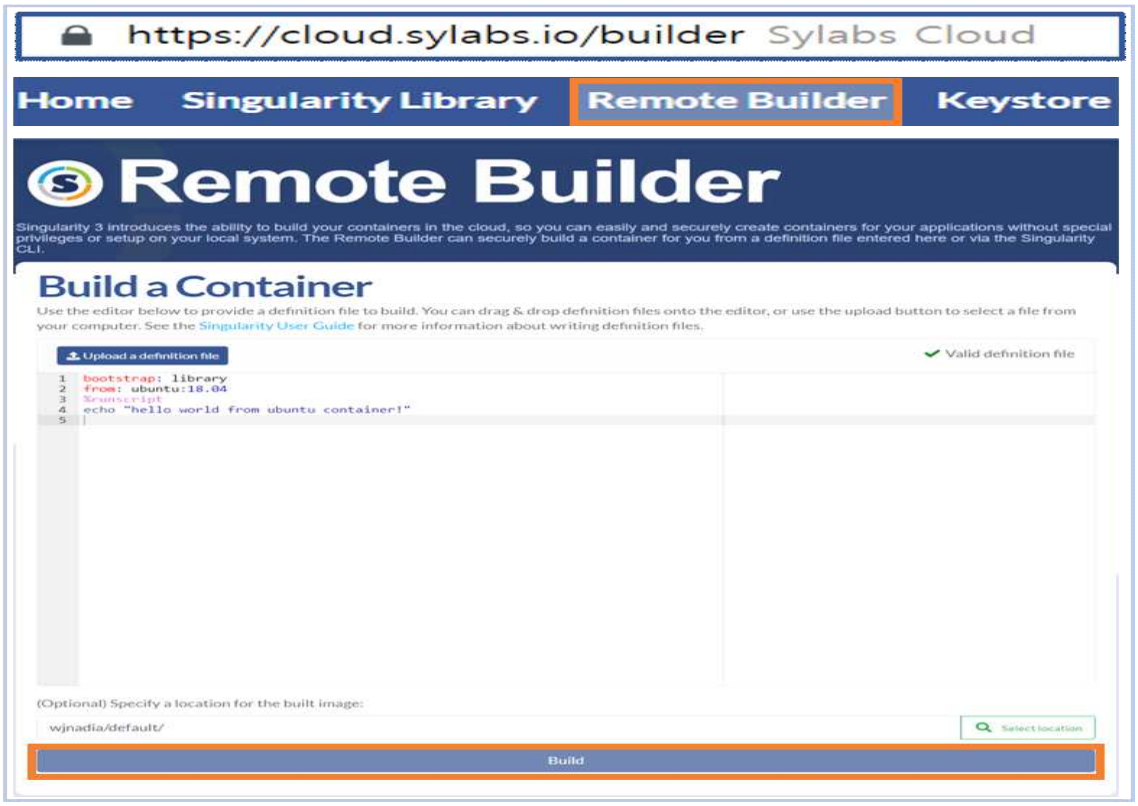
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MDg4NzY5MDQsImlhdCI6MTYwNjI4NDkwNCwiaXNzIjoiaHR0cHM6Ly9hdXRoLnN5bGFicy5pby90b2tlbiIsImp0aSI6IjVmYmRmNjY4NTJiN2I1NjMzZWFlNGVlMiIsInN1YiI6IjVkZjIyNzQ2ZjFhM2YyZGUwNmFlYTFmOSJ9.KxV1jd0O_8nWB5_xLVgM87hrGqWtkcgl8k9daq5JrN2qy-xeqmUlNVu2R-vRJWovK9k-ojfNZZKJHMH2dOhZMWpkqkPEv5HGZE2A59OyVPRVBrEpkwFYI1OseyYO7yGCn6vtX_UGkp6KB2ewsz4kK33L69DW8-Xrjq9xxRf3S3Gm-yTw2yy-fYUmWyh8st1U08LUXHJEIzkCGO-b5JXNZd0kfA8_AqftGDsQtE6SSmlqDT37L8378WkgCZE5HemjjTkodT1q1wK9AnPbUjEYix6BxN3wP5B0qI6PqTogz8Y5LfWgdJxN77LCzvFsG4wthKmqggg7sg8talkW5JMjHg

[참조2]

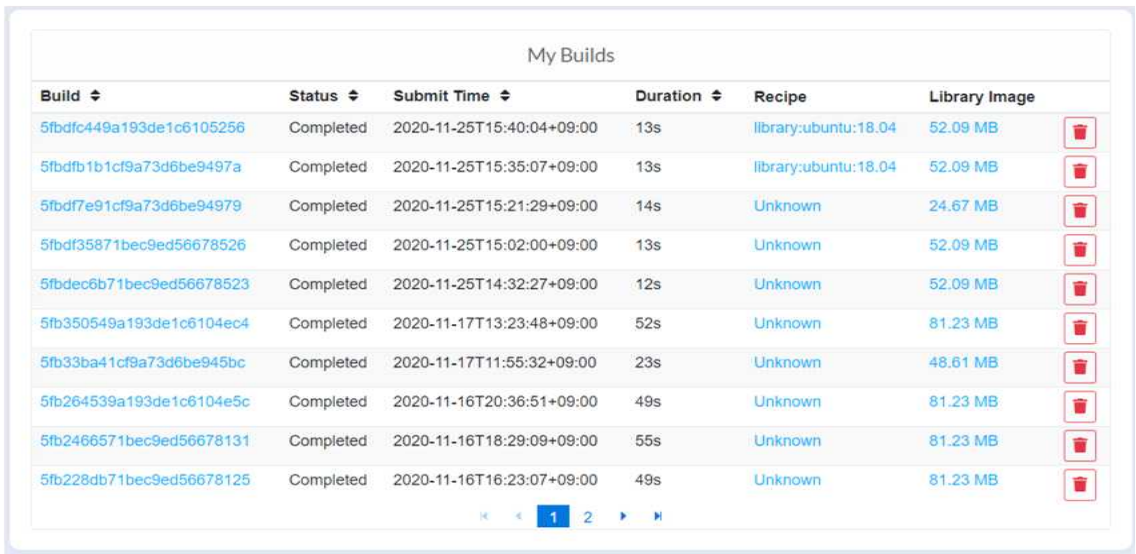
웹 브라우저에서 리모트 빌더에 의한 싱귤레러티 컨테이너 빌드하기

[Sylabs Cloud 바로가기]

① 웹 브라우저에서 컨테이너 이미지 빌드하기



② 빌드한 컨테이너 이미지 목록 보기



Build	Status	Submit Time	Duration	Recipe	Library Image
5fbdfc449a193de1c6105256	Completed	2020-11-25T15:40:04+09:00	13s	library:ubuntu:18.04	52.09 MB
5fbdfb1b1cf9a73d6be9497a	Completed	2020-11-25T15:35:07+09:00	13s	library:ubuntu:18.04	52.09 MB
5fbdf7e91cf9a73d6be94979	Completed	2020-11-25T15:21:29+09:00	14s	Unknown	24.67 MB
5fbdf35871bec9ed56678526	Completed	2020-11-25T15:02:00+09:00	13s	Unknown	52.09 MB
5fbdec6b71bec9ed56678523	Completed	2020-11-25T14:32:27+09:00	12s	Unknown	52.09 MB
5fb350549a193de1c6104ec4	Completed	2020-11-17T13:23:48+09:00	52s	Unknown	81.23 MB
5fb33ba41cf9a73d6be945bc	Completed	2020-11-17T11:55:32+09:00	23s	Unknown	48.61 MB
5fb264539a193de1c6104e5c	Completed	2020-11-16T20:36:51+09:00	49s	Unknown	81.23 MB
5fb2466571bec9ed56678131	Completed	2020-11-16T18:29:09+09:00	55s	Unknown	81.23 MB
5fb228db71bec9ed56678125	Completed	2020-11-16T16:23:07+09:00	49s	Unknown	81.23 MB

[참조3]
병렬 학습 프로그램 실행 예제

- 아래 예제는 싱귤레러티 컨테이너에서 pytorch 혹은 keras(tensorflow)로 작성된 resnet50 모델을 사용하여 imagenet 이미지 분류를 위한 병렬 학습 실행을 사용자가 직접 따라해 볼 수 있도록 구성됨

■ 병렬 학습 작업 스크립트 경로 : /apps/applications/singularity_images/examples

■ 컨테이너 이미지 디렉터리 경로 : /apps/applications/singularity_images/ngc

■ 병렬 학습 예제 프로그램 경로
- pytorch 프로그램
  (단일노드) /apps/applications/singularity_images/examples/pytorch/resnet50v1.5
  (멀티노드-horovod) /apps/applications/singularity_images/examples/horovod/examples/pytorch
- keras(Tensorflow) 프로그램
  (멀티노드-horovod) /apps/applications/singularity_images/examples/horovod/examples/keras

■ imagent 이미지 데이터 경로
- (학습 데이터) /apps/applications/singularity_images/imagenet/train
- (검증 데이터) /apps/applications/singularity_images/imagenet/val

1) /apps/applications/singularity_images/examples 디렉터리에서 아래 작업 스크립트 파일을 사용자 작업 디렉터리로 복사함

```
[a1234b5@glogin01]$ cp /apps/applications/singularity_images/examples/*.sh /scratch/ID/work/
```

2) STATE가 idle 상태인 계산 노드가 있는 파티션을 확인함

아래 예제에서는 cas_v100nv_8, cas_v100nv_4, cas_v100_4, cas_v100_2 등의 파티션에 가용 계산노드가 존재함

```
[a1234b5@glogin01]$ sinfo
```

```
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
jupyter         up 2-00:00:00      2  alloc jupyter[03-04]
jupyter         up 2-00:00:00      2   idle jupyter[01-02]
cas_v100nv_8    up 2-00:00:00      3  alloc gpu[01-03]
cas_v100nv_4    up 2-00:00:00      1    mix gpu07
cas_v100nv_4    up 2-00:00:00      2  alloc gpu[06,08]
cas_v100_4      up 2-00:00:00      4    mix gpu[11,16,19,21]
cas_v100_4      up 2-00:00:00     10  alloc gpu[10,12-15,17-18,20,22-23]
cas_v100_2      up 2-00:00:00      1   plnd gpu25
cas_v100_2      up 2-00:00:00      1  alloc gpu26
debug           up    2:00:00      2   idle gpu[27-28]
skl             up 2-00:00:00      1    mix skl03
skl             up 2-00:00:00      6  alloc skl[01-02,04-05,07-08]
skl             up 2-00:00:00      2   idle skl[09-10]
bigmem          up 2-00:00:00      2    mix bigmem[01,03]
bigmem          up 2-00:00:00      1   idle bigmem02
```

3) 작업 스크립트 파일에서 작업명(-J), wall_time(--time), 작업 큐(-p), Application이름(--comment), 계산노드 자원 요구량(--nodes, --ntasks-per-node, --gres) 등의 스케줄러 옵션과 학습 프로그램의 파라미터를 변경함

```
[a1234b5@glogin01]$ vi 01.pytorch.sh
#!/bin/sh
#SBATCH -J pytorch #job name
#SBATCH --time=24:00:00 # walltime
#SBATCH --comment=pytorch # application name
#SBATCH -p cas_v100_4 # partition name (queue or class)
#SBATCH --nodes=1 # number of nodes
#SBATCH --ntasks-per-node=2 # number of tasks per node
#SBATCH --cpus-per-task=10 # number of cpus per task
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

## Training Resnet-50(Pytorch) for image classification on single node & multi GPUs
Base=/apps/applications/singularity_images
module load ngc/pytorch:22.03-py3

python $Base/examples/pytorch/resnet50v1.5/multiproc.py --nproc_per_node 2 $Base/examples/pytorch/resnet50v1.5/main.py $Base/imagenet \
--data-backend dali-gpu --raport-file report.json -j2 --arch resnet50 -c fanin --label-smoothing 0.1 -b 128 --epochs 50
```

4) 스케줄러에 작업을 제출함

```
[a1234b5@glogin01]$ sbatch 01.pytorch.sh
Submitted batch job 99982
```

5) 스케줄러에 의해 할당된 계산 노드를 확인함

```
[a1234b5@glogin01]$ squque -u a1234b5
 JOBID PARTITION NAME USER STATE TIME TIME_LIMI NODES NODELIST(REASON)
 99982 cas_v100_2 pytorch a1234b5 RUNNING 10:13 24:00:00 1 gpu41
```

6) 스케줄러에 의해 생성되는 로그 파일을 모니터링함

```
[a1234b5@glogin01]$ tail -f pytorch_99982.out
 or
[a1234b5@glogin01]$ tail -f pytorch_99982.err
```

```
98
63
RUNNING EPOCHS FROM 0 TO 50
DLL 2022-03-10 21:51:28.935912 - Epoch: 0 Iteration: 10  train.loss : 8.33368  train.total_ips : 245.54 img/s
DLL 2022-03-10 21:51:45.562542 - Epoch: 0 Iteration: 20  train.loss : 8.52414  train.total_ips : 154.29 img/s
DLL 2022-03-10 21:52:03.233860 - Epoch: 0 Iteration: 30  train.loss : 7.68996  train.total_ips : 148.80 img/s
DLL 2022-03-10 21:52:19.457961 - Epoch: 0 Iteration: 40  train.loss : 7.08698  train.total_ips : 158.21 img/s
DLL 2022-03-10 21:52:36.266374 - Epoch: 0 Iteration: 50  train.loss : 6.99963  train.total_ips : 152.48 img/s
DLL 2022-03-10 21:52:53.962328 - Epoch: 0 Iteration: 60  train.loss : 6.96700  train.total_ips : 145.37 img/s
DLL 2022-03-10 21:53:10.840057 - Epoch: 0 Iteration: 70  train.loss : 6.94762  train.total_ips : 152.14 img/s
DLL 2022-03-10 21:53:27.776470 - Epoch: 0 Iteration: 80  train.loss : 6.96804  train.total_ips : 151.38 img/s
```

7) 스케줄러에 의해 할당된 계산 노드에서 학습 프로세스 및 GPU 활용 현황을 모니터링함

```
[a1234b5@glogin01]$ ssh gpu41
[a1234b5@gpu41]$ module load nvtop
[a1234b5@gpu41]$ nvtop
```

• 작업 스크립트

1) pytorch 단일 노드 병렬 학습(01.pytorch.sh)

```
#!/bin/sh
#SBATCH -J pytorch #job name
#SBATCH --time=24:00:00 # walltime
#SBATCH --comment=pytorch # application name
#SBATCH -p cas_v100_4 # partition name (queue or class)
#SBATCH --nodes=1 # number of nodes
#SBATCH --ntasks-per-node=2 # number of tasks per node
#SBATCH --cpus-per-task=10 # number of cpus per task
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

## Training Resnet-50(Pytorch) for image classification on single node & multi GPUs
Base=/apps/applications/singularity_images
module load ngc/pytorch:22.03-py3

python $Base/examples/pytorch/resnet50v1.5/multiproc.py --nproc_per_node 2 $Base/examples/pytorch/resnet50v1.5/main.py $Base/imagenet \
--data-backend dali-gpu --raport-file report.json -j2 --arch resnet50 -c fanin --label-smoothing 0.1 -b 128 --epochs 50
```

※ 1노드 점유, 노드당 2 타스크, 타스크당 10 CPUs, 노드당 2GPU 사용

2) pytorch_horovod 멀티 노드 병렬 학습(02.pytorch_horovod.sh)

```
#!/bin/sh
#SBATCH -J pytorch_horovod # job name
#SBATCH --time=24:00:00 # walltime
#SBATCH --comment=pytorch # application name
#SBATCH -p cas_v100_4 # partition name (queue or class)
#SBATCH --nodes=2 # the number of nodes
#SBATCH --ntasks-per-node=2 # number of tasks per node
#SBATCH --cpus-per-task=10 # number of cpus per task
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

## Training Resnet-50(Pytorch horovod) for image classification on multi nodes & multi GPUs
Base=/apps/applications/singularity_images
module load ngc/pytorch:22.03-py3

mpirun_wrapper \
python $Base/examples/horovod/examples/pytorch/pytorch_imagenet_resnet50.py \
--batch-size=128 --epochs=50
```

※ 2노드 점유, 노드당 2 MPI 타스크, 타스크당 10 CPUs, 노드당 2GPU 사용

3) keras(tensorflow)_horovod 멀티 노드 병렬 학습(03.keras_horovod.sh)

```
#!/bin/sh
#SBATCH -J keras_horovod # job name
#SBATCH --time=24:00:00 # walltime
#SBATCH --comment=tensorflow # application name
#SBATCH -p cas_v100_4 # partition name (queue or class)
#SBATCH --nodes=2 # the number of nodes
#SBATCH --ntasks-per-node=2 # number of tasks per node
#SBATCH --cpus-per-task=10 # number of cpus per task
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --gres=gpu:2 # number of GPUs per node

## Training Resnet-50(Keras horovod) for image classification on multi nodes & multi GPUS
Base=/apps/applications/singularity_images/examples
module load ngc/tensorflow:22.03-tf1-py3

mpirun_wrapper python $Base/horovod/examples/keras/keras_imagenet_resnet50.py \
--batch-size=128 --epochs=50
```

※ 2노드 점유, 노드당 2 MPI 타스크, 타스크당 10 CPUs, 노드당 2GPU 사용

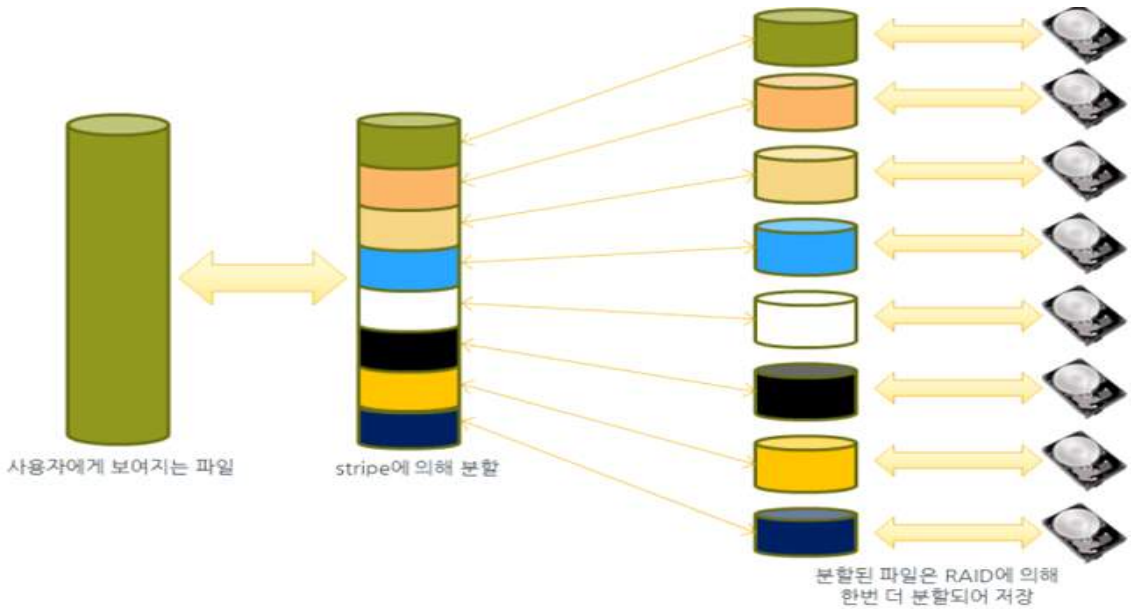
Ⅲ-4. Lustre stripe

가. Neuron Lustre Striping 기본 설정

Neuron Lustre File system은 File Striping을 지원하며, 이를 위해 복수 개의 OST(Object Storage Target, 즉 물리적으로 분산되어 있는 여러 디스크)에 하나의 파일을 분산시켜 저장함으로써 병목을 줄이고 I/O 성능을 향상시킬 수 있다. 특히, Lustre 2.10부터 지원되는 PFL(Progressive File Layout)이 /scratch 파일시스템에 적용되어 있다. 이 기능은 사용자가 별도의 striping 설정 없이 파일의 크기에 따라 stripe-count 개수가 자동으로 적용되어 I/O의 성능을 향상시킬 수 있다. Neuron 파일 시스템의 striping 설정은 아래와 같다.

		stripe-count	stripe-size
/home01		1	1MB
/apps		1	1MB
/scratch (PFL)	file < 4MB	1	1MB
	4MB < file < 512MB	2	1MB
	512MB < file < 1G	4	1MB
	1G < file < 10G	8	1MB
	10G < file < 100G	16	1MB
	file > 100G	32	1MB

나. Lustre Striping 개념



Lustre는 각 OST별로 자료를 분할하여 대용량 파일에 대한 I/O 성능을 최대화할 수 있으며, 병렬화가 유효한 최대 분할 수는 OST 숫자와 같다. 단일 파일 역시 위 그림과 같이 Lustre Striping 기능을 사용하여 OST에 병렬로 저장함.

다. Stripe 설정 및 확인

```
$ lfs setstripe [--stripe-size|-s size] [--stripe-count|-c count] filename|dirname
```

- 파일 또는 디렉터리에 striping 설정을 적용시키는 명령어. 위 명령으로 생성된 파일이나 위 명령이 적용된 디렉터리에서 생성되는 모든 파일은 striping 설정 적용
  - --stripe-size
    - 각 OST에 저장할 데이터의 크기를 설정
    - 지정된 크기만큼 저장하면 다음 OST에 데이터를 저장
    - 기본값은 1MB이며 stripe_size를 0으로 설정하면 기본값을 사용함
    - stripe_size는 반드시 64KB의 배수로 설정해야 하며 4GB보다 작아야 함
  - --stripe-count
    - Striping에 사용할 OST 개수를 설정
    - 기본값은 1이며 stripe_count를 0으로 설정하면 기본값을 사용
    - stripe_count가 -1이면 가능한 모든 OST들을 사용

```
$ lfs setstripe [--stripe-size|-s size] [--stripe-count|-c count] filename|dirname
```

※ 파일 또는 디렉터리에 적용된 striping 설정값을 확인하는 명령어

라. 권장사항 및 팁

- 작업 스크립트 내에서 모델의 결과 파일이 저장될 디렉터리에 대해 setstripe를 지정하면, 이후 생성되는 하위 디렉터리 및 파일은 모두 해당 설정값 상속
- --stripe-count는 파일 사이즈가 1GB 이상인 파일에 대해 4로 설정 시 대부분 성능 향상. 더 큰 값 사용 시 테스트 필요
- --stripe-size는 파일 사이즈가 수 TB 이상인 파일인 경우에만 유효하므로 대부분 default 값을 사용해도 문제없음

Ⅲ-5. 뉴론 Jupyter

가. Jupyter 개요

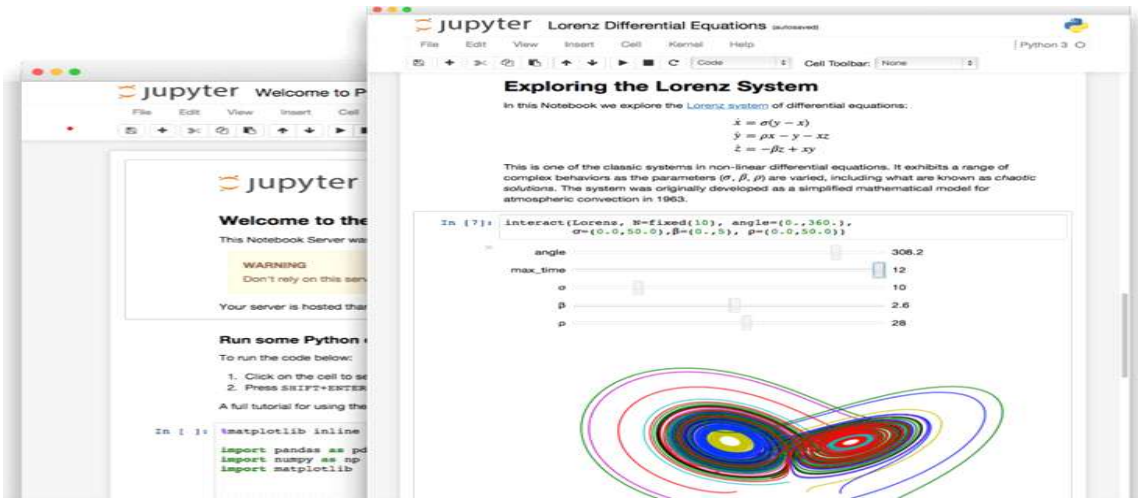
1. JupyterHub

- JupyterHub란 멀티 사용자 환경에서 주피터 랩/노트북을 사용할 수 있는 오픈소스 소프트웨어를 뜻한다.
- JupyterHub는 다양한 환경 (JupyterLab, Notebook, RStudio, Nteract 등)을 지원할 뿐만 아니라 인증 서버 (OAuth, LDAP, GitHub 등) 및 배치 스케줄러와도 (PBSPro, Slurm, LSF 등) 유연하게 연동 가능하다.
- JupyterHub는 컨테이너 관리 플랫폼인 Kubernetes와도 연동이 쉬워 컨테이너 기반의 클라우드 환경에 쉽게 연동 가능하다.

※ Neuron 기반 JupyterHub는 5호기 Bright LDAP, OTP 인증 기능을 추가하였고 Slurm 배치 스케줄러와 연동하여 자원을 할당하여 Jupyter 실행하고 현재 default로 Jupyter Notebook을 제공하고 추가로 JupyterLab 제공한다.

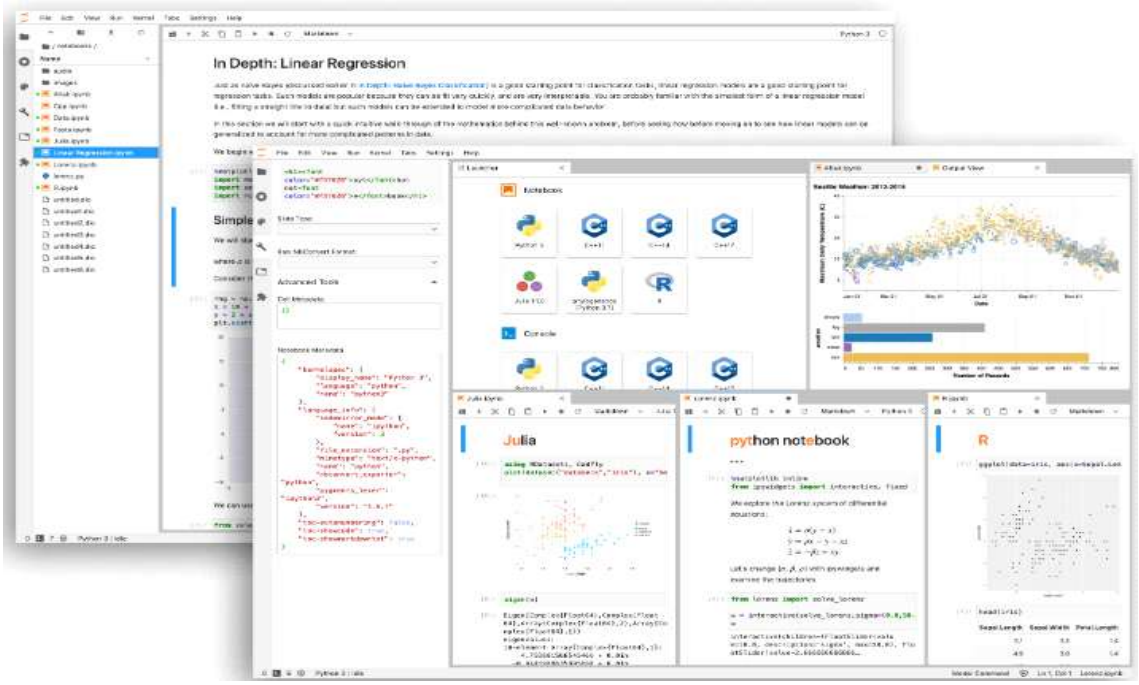
2. Jupyter Notebook

- Jupyter Notebook은 웹 기반의 오픈소스 어플리케이션으로 프로그래머들에게 문서 생성, 코드 생성 및 실행, 수학적 라이브러리를 사용한 데이터 시각화, 통계 모델링, 머신러닝/딥러닝 프로그래밍에 사용한다.
- 40여 개의 프로그래밍 언어 즉 Python, R, Julia, Scala등을 지원한다.
- 프로그래밍 언어로 작성한 코드는 HTML, 이미지, 동영상 파일, LaTeX 등 다양한 타입으로 변환 가능하다.
- Apache Spark, Pandas, Scikit-learn, ggplot2, Tensorflow 등 다양한 툴/라이브러리들과 연동 가능하다.



3. JupyterLab

- JupyterLab은 Jupyter Notebook 인터페이스에 사용자 편의를 위한 기능들을 추가하여 확장 가능한 모듈로 구성된다.
- Jupyter Notebook과 달리 하나의 작업 화면에 Tabs와 Splitters를 사용하여 여러 개의 도큐먼트 또는 다른 기능을 제공한다.



나. 스크립트를 통한 Jupyter 실행

1. 개요

- **로그인 노드 접속**

```
$ ssh [사용자ID]@neuron.ksc.re.kr
```

- **(필수) Jupyter 관련 패키지 설치 스크립트 실행**

```
$ sh /apps/jupyter/kisti_conda_jupyter.sh
```

※ notebook conda env 생성 및 jupyter notebook, jupyterlab, cudatoolkit 11.6, cudnn 패키지 설치

- **(옵션) 필요에 따라 기타 패키지 자동 설치 스크립트 실행 혹은 직접 실행**

※ notebook conda env 활성화 : **$ conda activate notebook**

1) tensorflow 작업 환경

```
(notebook) $ sh  /apps/jupyter/kisti_conda_tensorflow.sh
```

2) pytorch 작업 환경

```
(notebook) $ sh  /apps/jupyter/kisti_conda_pytorch.sh
```

※ 최초 한 번만 실행하며 환경설정이 완료되면 즉시 웹 페이지 접속하여 (다. JupyterLab 사용 방법) JupyterLab/Notebook을 사용 가능하다.

2. 스크립트 실행

- 터미널로 로그인 노드(neuron.ksc.re.kr)에 접속하여 다음 스크립트 /apps/jupyter/kisti_conda_jupyter.sh를 실행한다.
- 스크립트를 실행하면 /scratch/[사용자ID]/.conda/envs 디렉터리에 notebook Conda 환경이 만들어지고 jupyterhub, jupyterLab, notebook 패키지들이 자동으로 설치되고 멀티 GPU 환경에 필요한 cudatoolkit=11.6과 cudnn이 설치된다.

※ 이 파일은 한 번만 실행하면 되고 그다음부터는 바로 웹 페이지 접속하여 사용 가능하다.

※ 실행파일은 공유 디렉터리에서 /apps/jupyter/kisti_conda_jupyter.sh로 바로 실행 가능하다.

※ 아래 테스트는 사용자 ID *a1113a01*로 진행하였다.

```
[a1113a01@glogin02 ~]$ sh /apps/jupyter/kisti_conda_jupyter.sh
… …
modified /home01/a1113a01/.bashrc
…prepare conda environment for jupyter user.
Exporting CONDA ENVS and PKGS PATH to bash File.
Downloading and Extracting Packages
################################################################################ | 100%
################################################################################ | 100%
################################################################################ | 100%
################################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: / WARNING conda.core.prefix_data:_load_single_record(167):
done
```

• shell을 다시 시작하고 base 환경 자동 활성화 기능을 꺼야 한다. (한 번만 실행)

```
[a1113a01@glogin01 ~]$ source ~/.bashrc
(base) [a1113a01@glogin01 ~]$ conda config --set auto_activate_base false
(base) [a1113a01@glogin01 ~]$ conda deactivate
```

※ base 환경 자동 활성화 기능을 false로 설정함으로 다음에 base 환경으로 자동 활성화되는 것을 방지한다. (만약 base 환경으로 활성화되지 않았으면 source ~/.bashrc 이후 바로 conda activate notebook 명령어를 실행)

• conda notebook 환경을 다음 명령어로 활성화한다.

```
[a1113a01@glogin01 ~]$ conda activate notebook
(notebook) [a1113a01@glogin01 ~]$
```

• Tensorflow(tensorboard 포함) 혹은 Pytorch 등의 사용을 원하는 사용자는 KISTI에서 제공하는 자동 설치 스크립트를 실행하여 설치할 수 있다.

※ 주의: 반드시 notebook 사용자 환경에서 실행해야 한다.

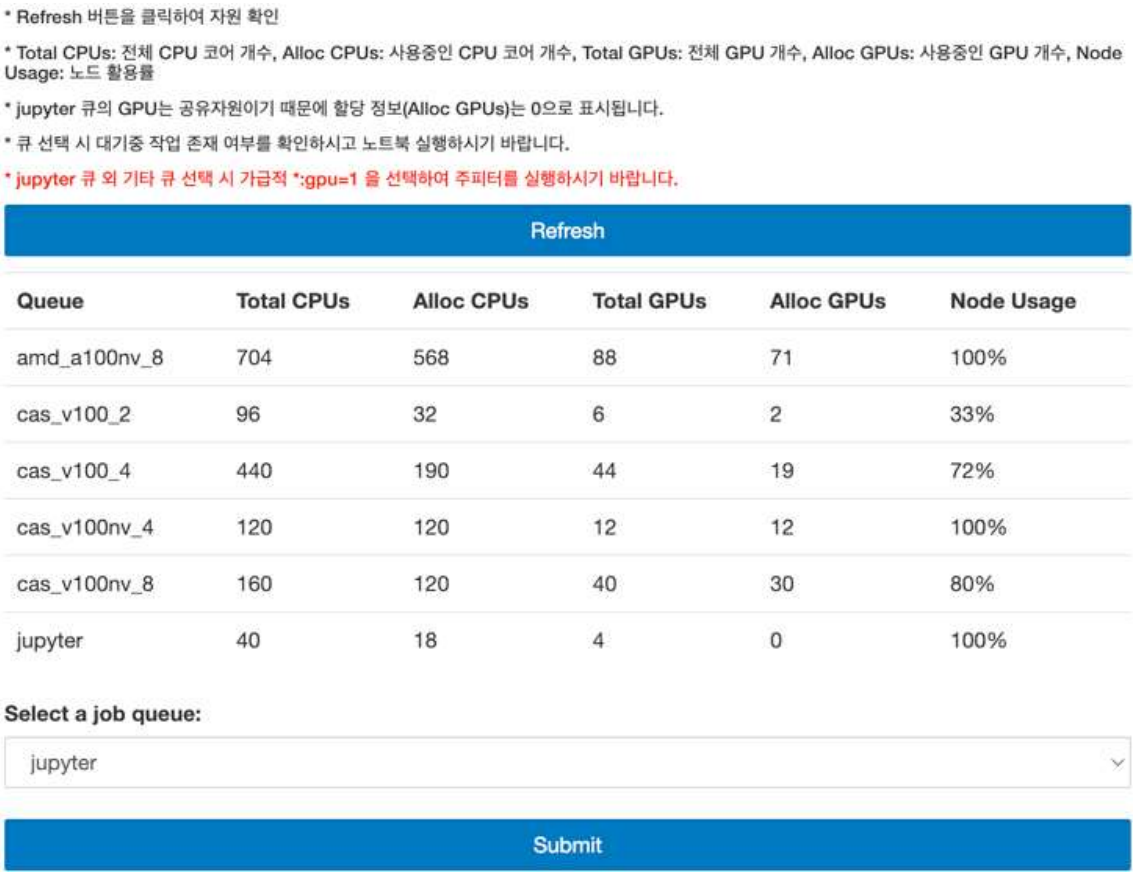
```
## tensorflow-gpu, tensorboard 패키지와 jupyter_notebook 및 jupyter_lab을 위한 tensorboard extenstion 설치
(notebook) 757% [a1113a01@glogin01 ~]$ sh /apps/jupyter/kisti_conda_tensorflow.sh (약 10분 소요)


## pytorch, torchvision, torchaudio 패키지 설치
(notebook) 757% [a1113a01@glogin01 ~]$ sh /apps/jupyter/kisti_conda_pytorch.sh (약 5분 소요)
```

※ 이제부터 사용자는 직접 웹에 접속하여 Jupyter 노트북을 사용할 수 있다. (여기까지 작업들은 한 번만 실행하면 됨)

3. JupyterHub 웹 페이지 접속

- https://jupyter.ksc.re.kr에 접속하여 신청받은 뉴런 계정, OTP, 비밀번호를 입력한다.

- 메인 화면에서 자원 사용 현황 확인 및 Refresh 버튼을 클릭하여 자원 사용 현황을 업데이트할 수 있다.



4. 큐 (Queue) 선택 및 Jupyter 실행

- Jupyter를 실행하기 전에 Refresh 버튼을 클릭하여 자원 현황을 확인
  - Total CPUs: 전체 CPU 코어 개수
  - Alloc CPUs: 사용 중인 CPU 코어 개수
  - Total GPUs: 전체 GPU 개수
  - Alloc GPUs: 사용 중인 GPU 개수
  - Node Usage: 노드 활용률
- Queue 정보 확인
  - jupyter queue (무료): 환경 설치, 전처리, 디버깅 용도
  - other queues (유료): 딥러닝/머신러닝 등 모델 실행 및 시각화 용도

※ jupyter queue는 현재 2개 노드로 최대 20개(노드당 10개) Jupyter Lab/Notebook 실행 가능함(여러 사용자가 노드의 CPU+GPU[v100] 공유)

※ jupyter queue의 GPU는 공유자원이기 때문에 할당 정보(Alloc GPUs)는 0으로 표시됩니다.

※ 큐 선택 시 대기 중 작업 존재 여부를 확인하시고 노트북 실행하시기 바랍니다.

※ jupyter 큐 외 기타 큐 선택 시 가급적 *:gpu=1을 선택하여 주피터를 실행하시기 바랍니다.

※ 유료 과금 정책은 기존 Neuron 시스템 과금 정책을 따르고 정보는 국가슈퍼컴퓨팅 홈페이지 요금 안내 페이지(https://www.ksc.re.kr/jwjg/gjbg/ygan)에서 확인 가능합니다.

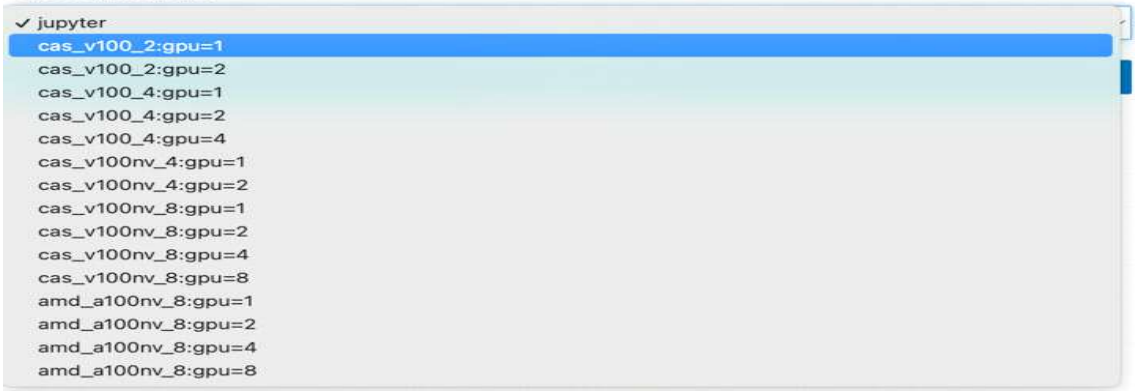
- Job queue에서 해당 queue를 선택하고 Submit 버튼을 클릭하여 Jupyter Notebook 실행(other queues로도 실행 가능하나, 다만 과금 발생함, 과금 정보는 KSC 홈페이지 Neuron 과금 정보 참고)

* Refresh 버튼을 클릭하여 자원 확인
* Total CPUs: 전체 CPU 코어 개수, Alloc CPUs: 사용중인 CPU 코어 개수, Total GPUs: 전체 GPU 개수, Alloc GPUs: 사용중인 GPU 개수, Node Usage: 노드 활용률
* jupyter 큐의 GPU는 공유자원이기 때문에 할당 정보(Alloc GPUs)는 0으로 표시됩니다.
* 큐 선택 시 대기중 작업 존재 여부를 확인하시고 노트북 실행하시기 바랍니다.
* jupyter 큐 외 기타 큐 선택 시 가급적 *:gpu=1 을 선택하여 주피터를 실행하시기 바랍니다.

Refresh

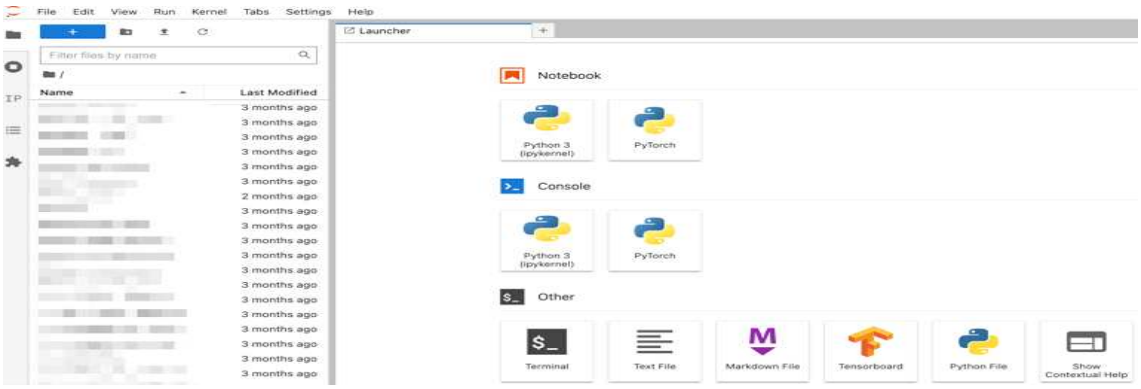
Queue	Total CPUs	Alloc CPUs	Total GPUs	Alloc GPUs	Node Usage
amd_a100nv_8	704	568	88	71	100%
cas_v100_2	96	32	6	2	33%
cas_v100_4	440	190	44	19	72%
cas_v100nv_4	120	120	12	12	100%
cas_v100nv_8	160	120	40	30	80%
jupyter	40	18	4	0	100%

Select a job queue:



- 다음과 같은 화면이 몇 초간 진행되면서 자원 할당이 진행된다.

- Default로 https://jupyter.ksc.re.kr/user/a1113a01/lab JupyterLab 화면이 실행된다.



다. JupyterLab 사용 방법

1) Jupyter 작업 환경

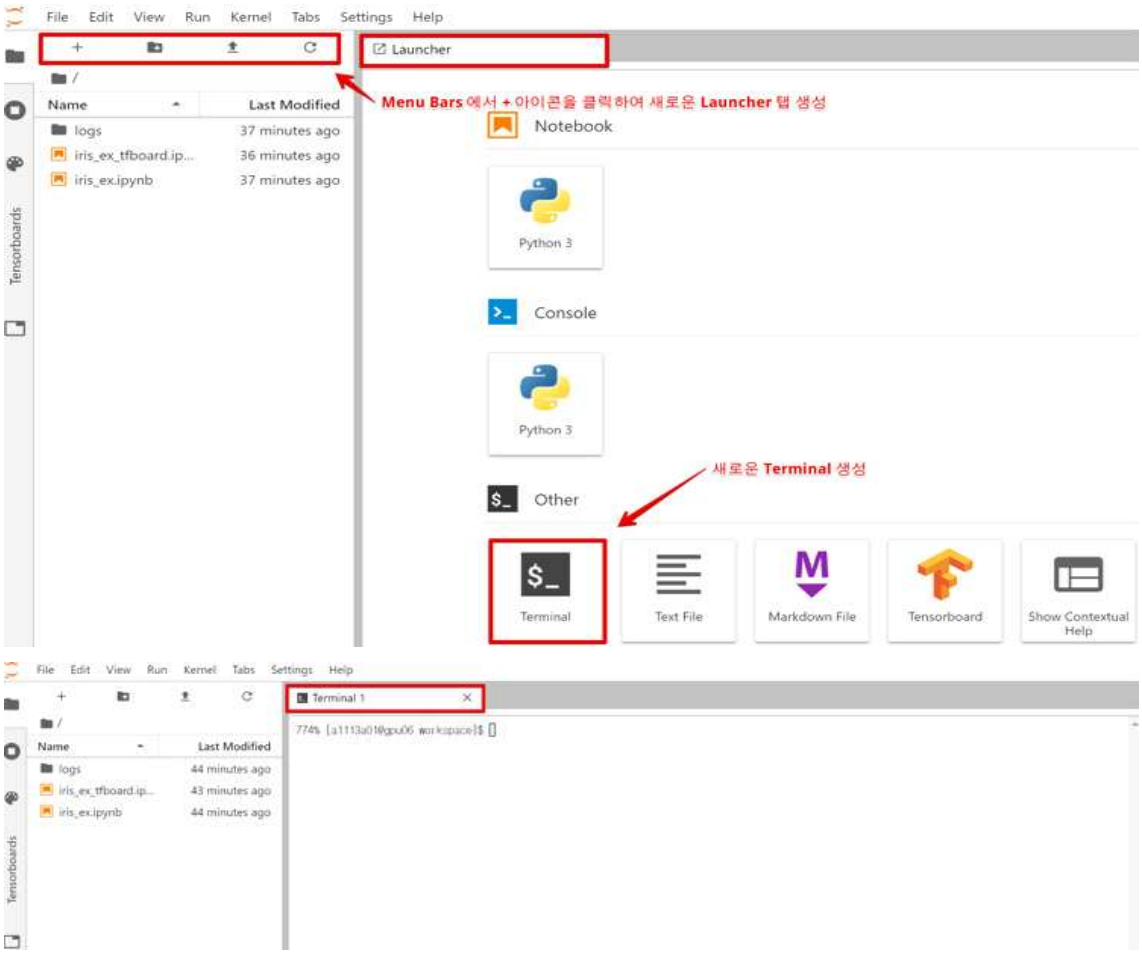
- Jupyter 환경 디렉터리: /scratch/[사용자ID]/.conda/envs/notebook
- 로그 저장 디렉터리: /scratch/[사용자ID]/log/작업ID.log
- 작업 파일 저장 디렉터리: /scratch/[사용자ID]/workspace/

※ 사용자는 본인이 필요로 하는 머신러닝/딥러닝 라이브러리들을 .../notebook conda 환경에 설치하기 때문에 기본 쿼터가 큰 /scratch/사용자ID/에 설치된다.(Jupyter 실행 후 발생하는 로그 파일도 /scratch/사용자ID에 저장)

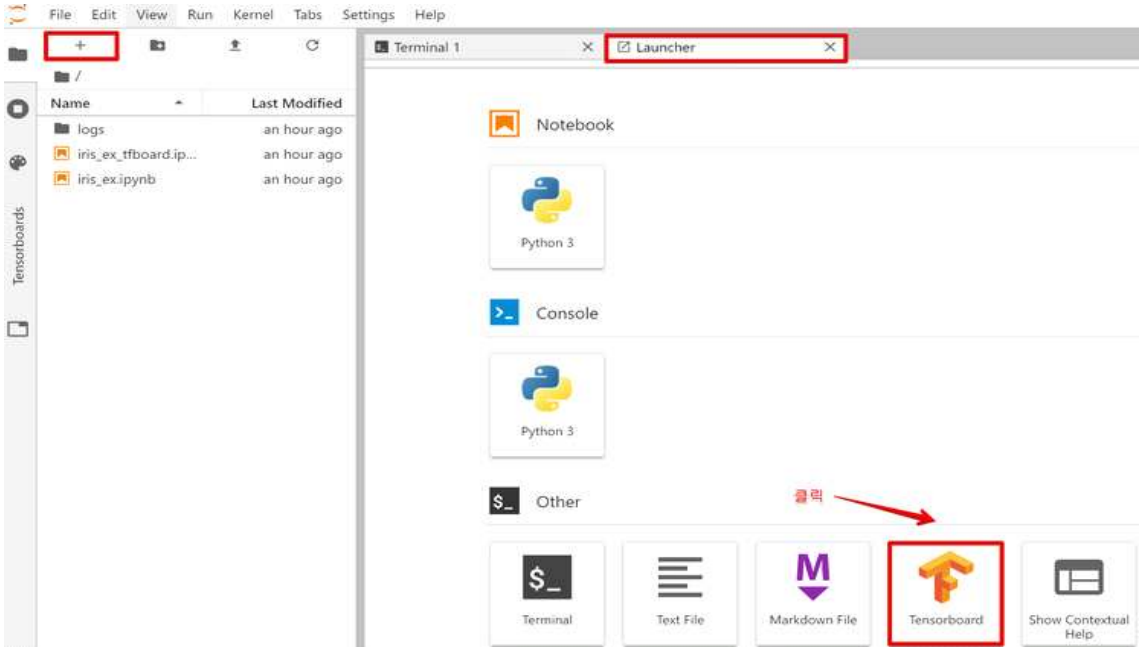
※ 사용자가 작성한 코드는 /scratch/사용자ID/에 저장된다.

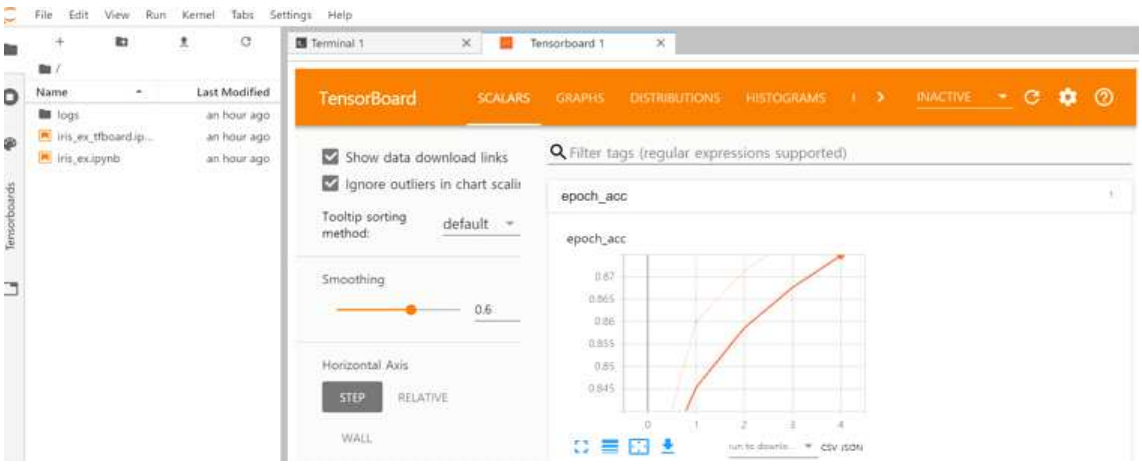
※ conda 환경 백업을 위한 conda 환경 내보내기 및 가져오기 관련 정보는 KISTI 홈페이지 소프트웨어 지침서에서 확인할 수 있다.

- Terminal 실행, Launcher 탭에서 Terminal 아이콘을 클릭한다.
  - Launcher 탭이 보이지 않을 경우 Menu Bars에서 + 아이콘을 클릭한다.



• Tensorboard 실행, Menu Bars → +아이콘 → Launcher → Tensorboard를 클릭한다.

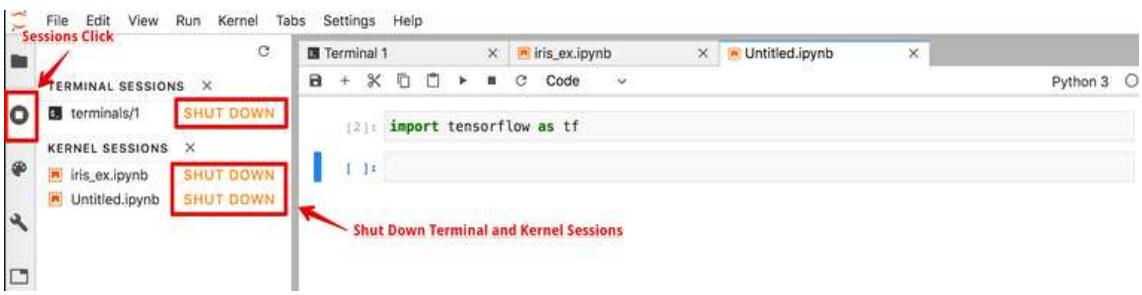




2) 실행 중인 세션 종료

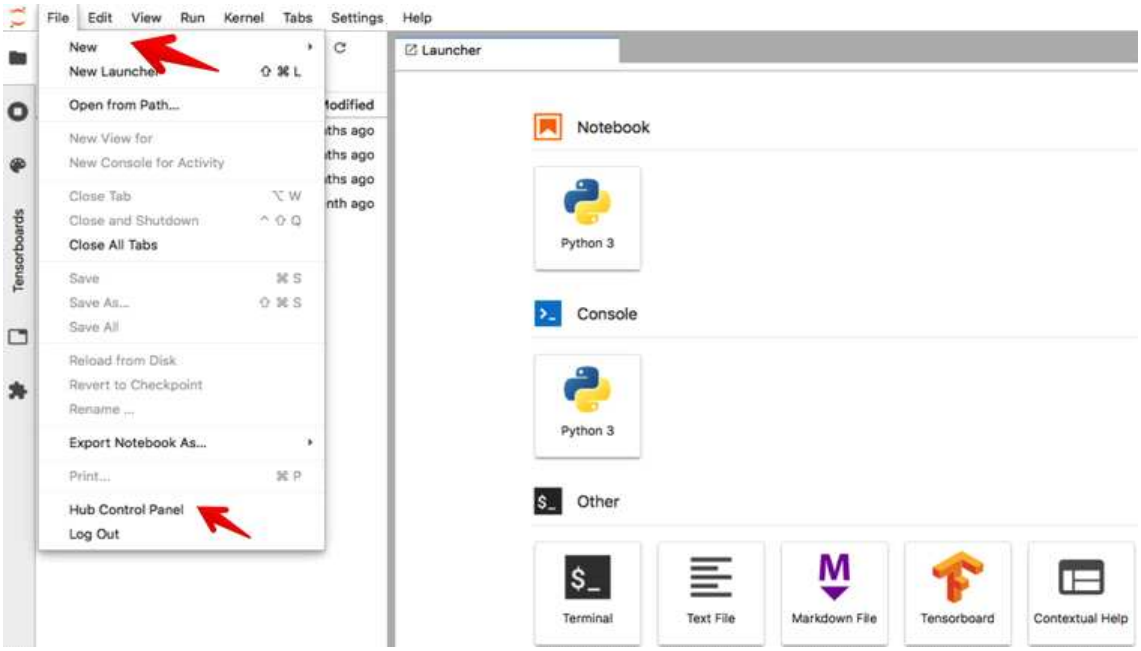
• 다음과 같이, Left Side Bar에서 Session 탭을 클릭하여 실행 중인 Terminal Sessions이나 Kernel Sessions들을 Shut Down 버튼을 클릭하여 종료한다.

※ 세션을 종료시키지 않고 JupyterHub 웹 페이지를 종료하는 경우, 다음 Jupyter 실행 시에도 그대로 남아있게 된다. (과금은 진행되지 않음)



3) Jupyter 종료

File -> Hub Control Panel -> Stop My Server



Neuron Jupyter 사용 안내

1. 서비스 목적

- GPU 기반 Jupyter Notebook/JupyterLab 사용 경험을 제공

* 사용방법은 Neuron Jupyter 사용자 지침서 참고 지침서 바로가기 : 뉴론->[별첨5]뉴론 Jupyter 사용법

2. 큐 정책

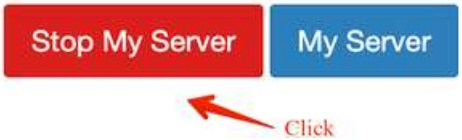
- Neuron 시스템의 큐와 동일한 큐를 사용
- GPU가 탑재된 계산노드에서만 jupyter 실행 가능(skl, bigmem 큐 등 제외)
- 작업 당 Jupyter 최대 사용 시간: 24시간
- 사용자 당 최대 Jupyter 작업 사용 개수: 1개
- 웹 페이지 logout시 자동 자원 반납

3. 과금

- jupyter큐는 환경 설치, 전처리, 디버깅 용도 : 무료 (ivy_v100_2, 2노드 최대 20명 동시 사용 가능)
- 그 밖의 큐는 딥러닝/머신러닝 등 모델 실행 및 시각화 용도: 유료

* 유료 과금 정보는 KSC 홈페이지-> 기술지원 -> 지침서 -> 하드웨어 -> 뉴론 -> 사용자 환경 -> 제공 시간 참고 -> 바로 가기

* Chrome, Safari, Firefox, Microsoft Edge 사용 권장

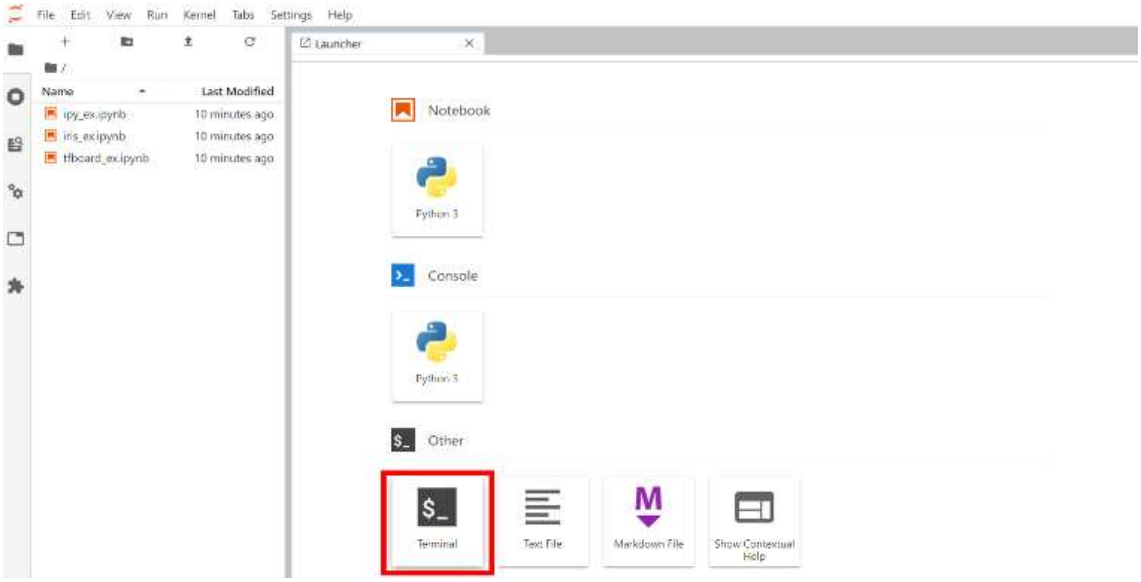


※ 홈페이지 Logout 시 실행 중인 Jupyter 및 세션들은 모두 자동으로 종료된다.

라. 머신러닝/딥러닝 예제 코드 실행하기

1. 예제 코드 실행에 필요한 라이브러리 설치

• Launcher에서 Terminal 클릭하여 머신러닝/딥러닝에 필요한 라이브러리를 설치한다.



• 터미널 환경에서 conda activate notebook 명령어로 notebook 환경을 활성화하고 notebook 환경에 필요한 라이브러리를 설치한다.

※ 반드시 notebook conda 환경에 설치해야 Jupyter 웹 화면에 적용된다.

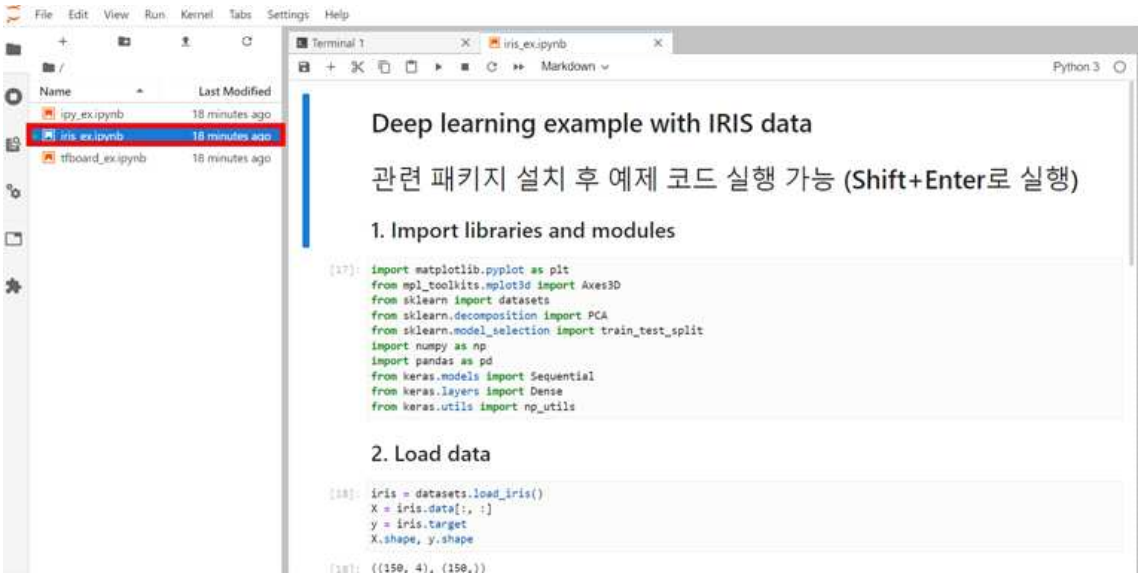
• notebook 환경에 사용자가 원하는 머신러닝/딥러닝 관련 라이브러리 설치 예시et

```
772% [a1113a01@gpu06 workspace]$ conda activate notebook
(notebook) 773% [a1113a01@gpu06 workspace]$ conda install -c conda-forge [conda 패키지 명]-y
```

2. 예제 코드 작성 및 실행

- 사용자 작업 디렉터리에서 예제 파일 iris_ex.ipynb를 클릭한다.
- 프로그램 편집/실행 창에서 Shift+Enter로 예제 코드를 실행한다.

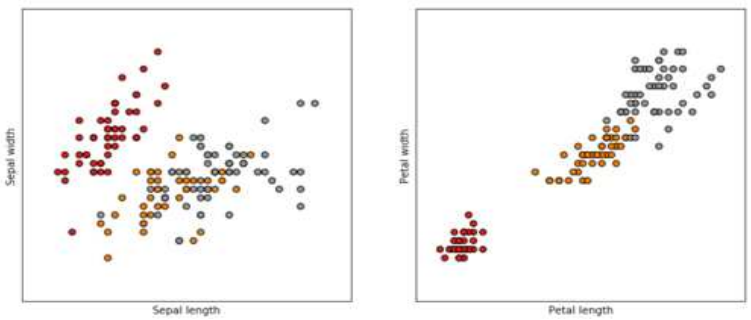
※ 실행 과정에 나오는 warning 들은 무시 가능하며, 동일 코드 재실행 시 warning 메시지 출력되지 않는다. (warning 내용은 코딩 시 버전에 따른 문법적 제시 안내)



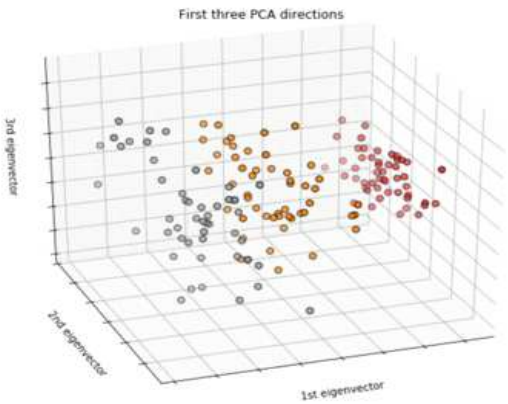
- matplotlib 라이브러리를 사용한 그래프 출력

3. Check data set

In [4]:
```
plt.figure(1, figsize=(12, 12))

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
ax1=plt.subplot(221)
# Plot the training points
ax1.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
ax1.set_xlabel('Sepal length')
ax1.set_ylabel('Sepal width')

ax1.set_xlim(x_min, x_max)
ax1.set_ylim(y_min, y_max)
ax1.set_xticks(())
ax1.set_yticks(())

ax2=plt.subplot(222)
x_min, x_max = X[:, 2].min() - .5, X[:, 2].max() + .5
y_min, y_max = X[:, 3].min() - .5, X[:, 3].max() + .5

# Plot the training points
ax2.scatter(X[:, 2], X[:, 3], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
ax2.set_xlabel('Petal length')
ax2.set_ylabel('Petal width')

ax2.set_xlim(x_min, x_max)
ax2.set_ylim(y_min, y_max)
ax2.set_xticks(())
ax2.set_yticks(())

plt.show()
```

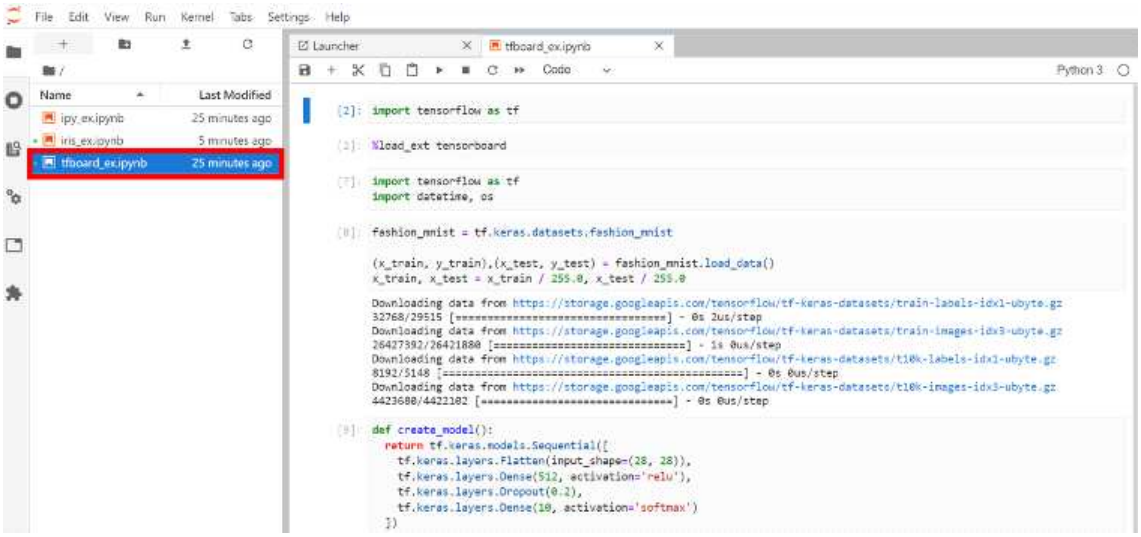


In [20]:
```
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```



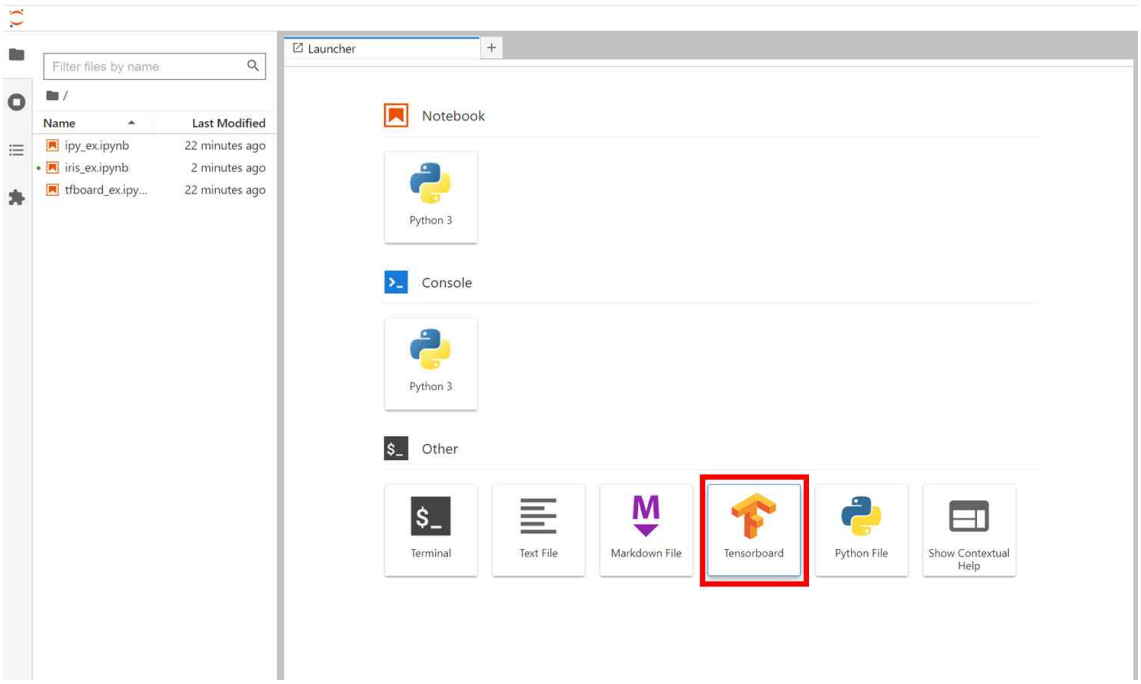
3. Tensorboard 실행

- Menu Bar -> Files에서 tfboard_ex.ipynb를 클릭한다.
- Shifter+Enter로 코드 실행한다.(약 1분 소요)

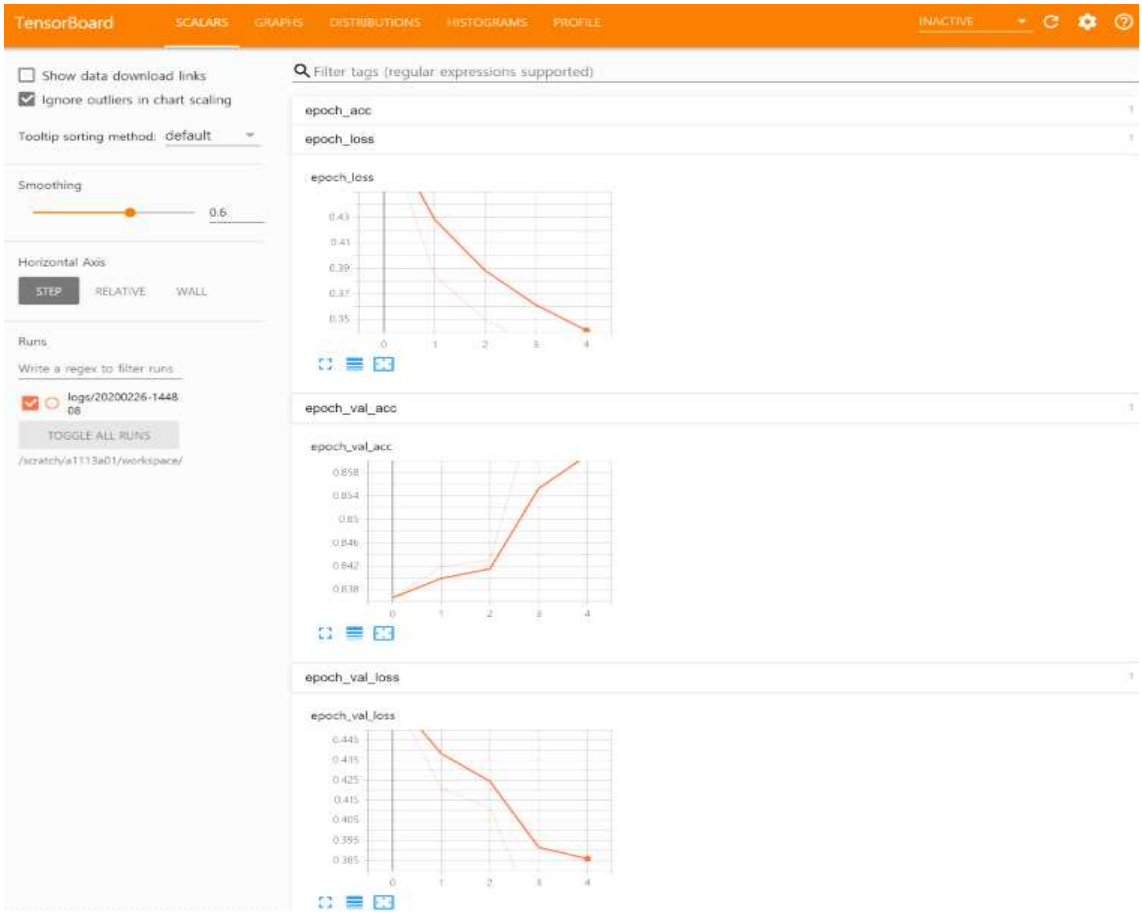


- Tensorboard 실행한다.

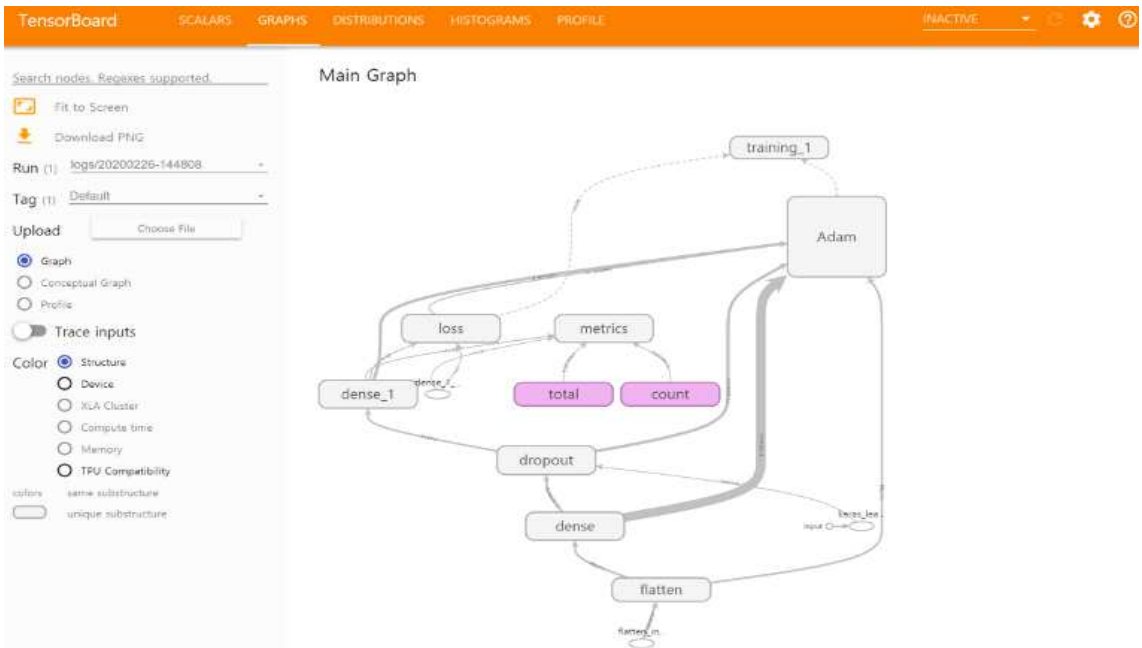
※ logs 폴더에 로그 데이터가 저장된다.



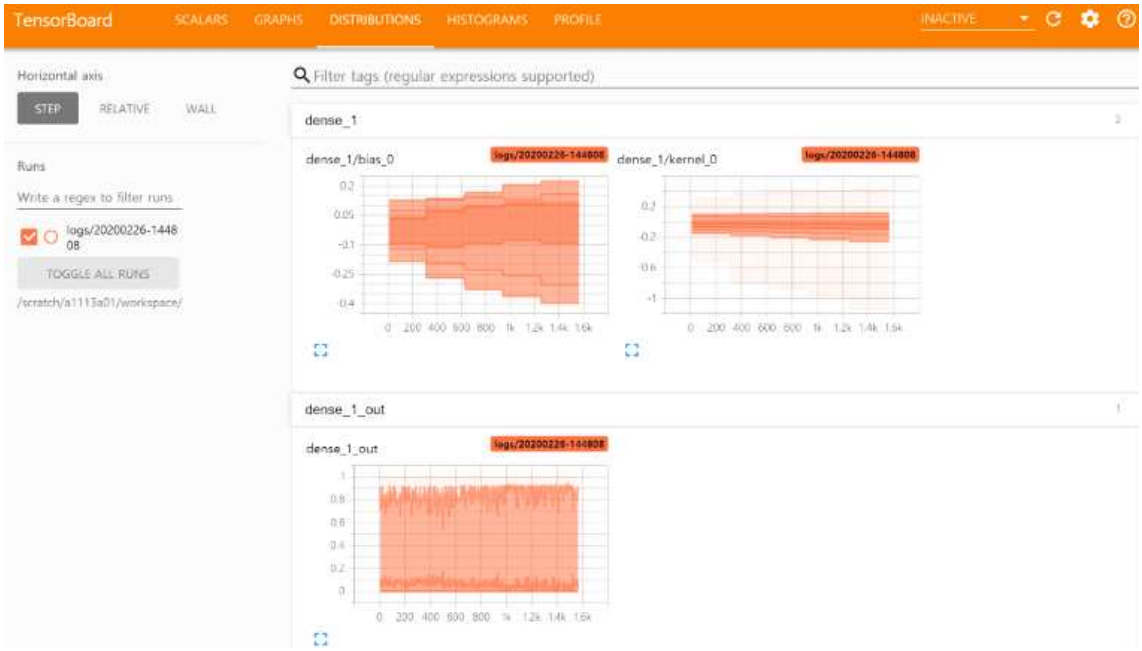
• TensorBoard -> Scalars



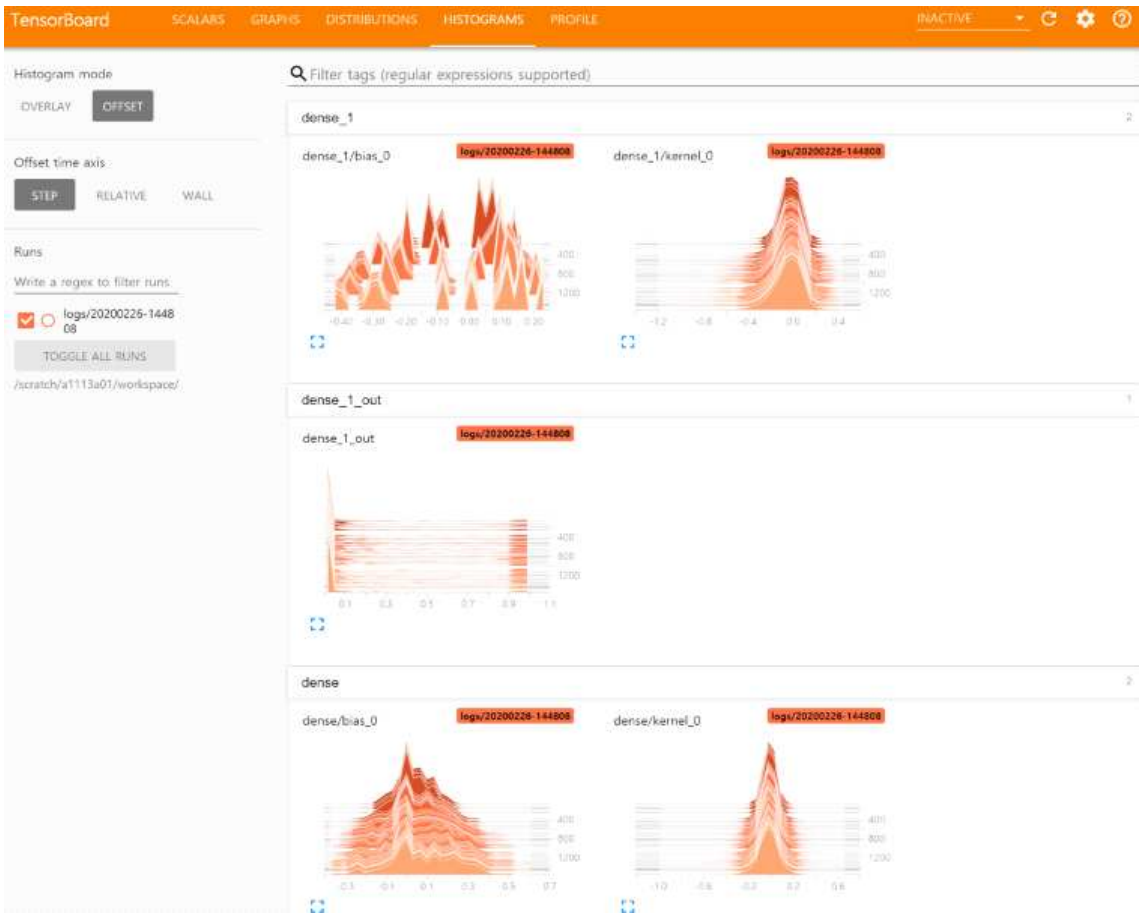
• TensorBoard -> Graphs



• TensorBoard -> Distributions

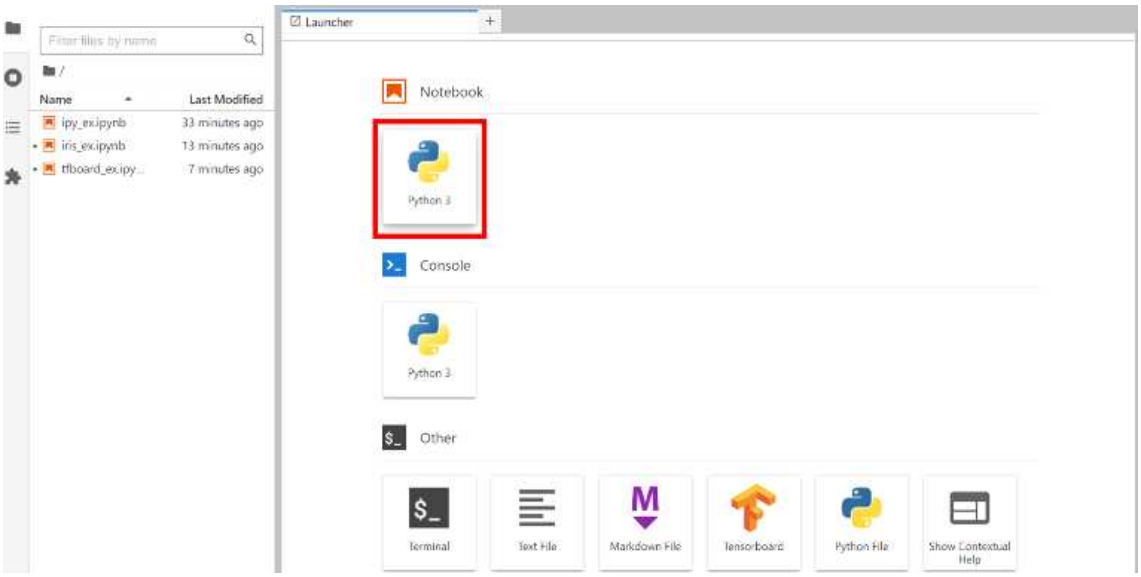


• TensorBoard -> Histograms

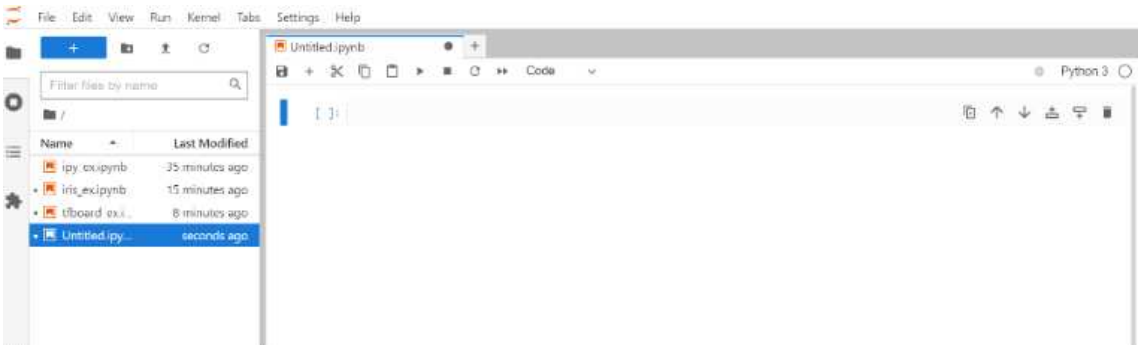


4. 새로운 Launcher 만들기 및 Python 코드 작성

- 아래와 같이 New -> Python 3 메뉴를 클릭하여 새로운 Python 코드의 작성이 가능하다.



- Python 3 커널을 사용할 수 있는 새로운 Jupyter Notebook Launcher가 실행된다.

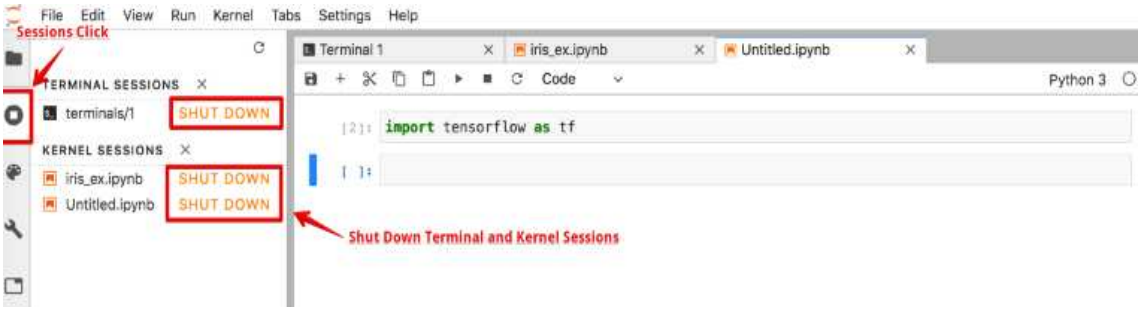


마. Jupyter 종료 방법

1. 실행 중인 세션 종료

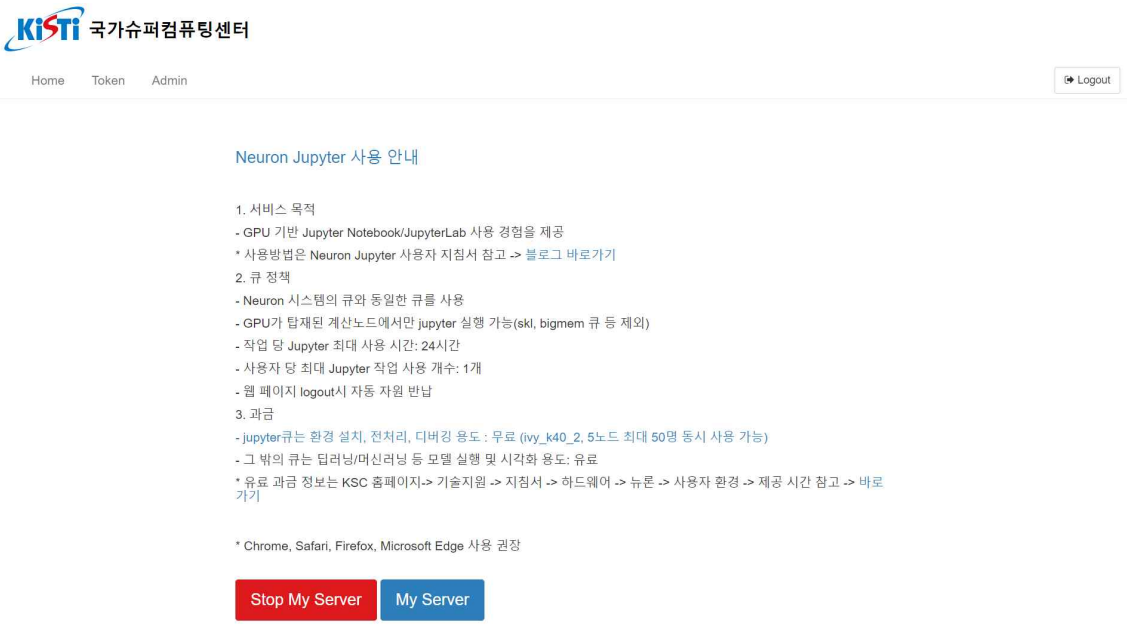
- 다음과 같이, Left Side Bar에서 Session 탭을 클릭하여 실행 중인 Terminal Sessions이나 Kernel Sessions들을 Shut Down 버튼을 클릭하여 종료한다.

※ 세션을 종료시키지 않고 JupyterHub 웹 페이지를 종료하는 경우, 다음 Jupyter 실행 시에도 그대로 남아있게 된다. (과금은 진행되지 않음)



2. Jupyter 종료

- (JupyterLab) Jupyter 사용이 끝나면 반드시 Jupyter를 종료시켜 자원을 반납해야 한다.
- File 메뉴에서 Hub Control Panel 클릭하여 Home 페이지로 와서 Stop My Server 클릭하여 자원을 반납할 수 있다.



바. Jupyter 환경 초기화 방법

- conda 가상 환경 notebook에 pip으로 설치할 경우 기존 conda install로 설치한 패키지들과 버전 충돌이 발생하여 Jupyter 노크북이 실행이 안될 경우 다음과 같은 명령어로 환경 초기화를 해줄 수 있다.
- 터미널로 로그인 노드에서 /apps/jupyter/reset_env.sh를 실행한다.
- 해당 스크립트를 실행하면 /scratch/[사용자ID]/.conda/envs 디렉터리에 만들어졌던 notebook 가상 환경에 설치되었던 모든 패키지들이 삭제되고 처음 jupyter 실행을 위한 기본 패키지들이 다시 설치된다.
- /sratch/[사용자ID]/workspace/에 데이터는 보존된다.

```
[a1113a01@glogin02 ~]$ sh /apps/jupyter/reset_env.sh

Remove all packages in environment /scratch/a1113a01/.conda/envs/notebook:

Preparing transaction: done

Verifying transaction: done

If you need another packages, you can run the installation scripts for some packages such as below !
1.(mandatary) conda activate notebook
2.(option)
  a. [tensorfow] sh /apps/jupyter/kisti_conda_tensorflow.sh
  b. [pytorch] sh /apps/jupyter/kisti_conda_pytorch.sh
  c. [etc] sh /apps/jupyter/kisti_conda_etc.sh

[a1113a01@glogin02 ~]$ conda activate notebook

## 필요에 따라 기타 패키지에 대한 자동 설치 스크립트 실행

(notebook)[a1113a01@glogin02 ~]$ /apps/jupyter/kisti_conda_tensorflow.sh
(notebook)[a1113a01@glogin02 ~]$ /apps/jupyter/kisti_conda_pytorch.sh
```

Ⅲ-6. Keras 기반 Multi GPU 사용법

Keras(케라스)는 파이썬으로 작성된 오픈 소스 신경망 라이브러리로, MXNet, Deeplearning4j, 텐서플로, Microsoft Cognitive Toolkit 또는 Theano 위에서 수행할 수 있는 High-level Neural Network API이다. NEURON의cas_v100_2, cas_v100nv_4, cas_v100nv_8, amd_a100nv_8 큐에는 한 노드에 2개, 4개, 8개의 GPU가 장착되어 있어, 단일노드를 이용할 때에도 GPU를 여러 대 사용하여 신경망 학습을 할 수 있는 환경이 구축되어 있다.

가. Multi-GPU 사용을 위한 코드 변경 및 작업 제출 방법

1. [from keras.utils import multi_gpu_model] 모듈 추가

```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import multi_gpu_model
import os
```

2. 코드 내 multi-gpu 사용 선언

```
# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
# multi-gpu
model = multi_gpu_model(model, gpus=2)
# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

※ 사용하고자 하는 GPU 개수만큼 gpus를 설정.(ex. cas_v100nv_4노드의 경우에는 gpus=4라고 설정)

3. 작업제출 스크립트

```
#!/bin/sh
#SBATCH -J keras
#SBATCH --time=24:00:00
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH -p cas_v100_4
#SBATCH --comment tensorflow
#SBATCH --gres=gpu:2
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 conda/tensorflow_2.4.1

srun python example.py
```

Ⅲ-7. Conda 기반 Horovod 설치 방법

Horovod는 고성능 분산 컴퓨팅 환경에서 노드 간 메시지 전달 및 통신관리를 위해 일반적인 표준 MPI 모델을 사용하며, Horovod의 MPI 구현은 표준 Tensorflow 분산 훈련 모델보다 간소화된 프로그래밍 모델을 제공한다. NEURON 시스템에서도 콘다 환경을 기반으로 멀티노드를 이용한 훈련 모델을 학습시키고자 한다면 다음과 같은 방법으로 설치 후 실행할 수 있다.

※ Horovod 사용법은 [별첨8] 참고.

가. Tensorflow-horovod 설치

1. 콘다 환경 생성

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1 cmake/3.16.9
$ conda create -n my_tensorflow
$ source activate my_tensorflow
(my_tensorflow) $
```

※ 자세한 콘다 사용 방법은 [별첨5] 참고

2. Tensorflow 설치 및 horovod 설치

```
(my_tensorflow) $ conda install tensorflow-gpu=2.0.0 tensorboard=2.0.0 tensorflow-estimator=2.0.0 python=3.7 cudnn cudatoolkit=10 nccl=2.8.3
(my_tensorflow) $ HOROVOD_WITH_MPI=1 HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_NCCL_LINK=SHARED HOROVOD_WITH_TENSORFLOW=1 \
pip install --no-cache-dir horovod==0.23.0
```

3. Horovod 설치 확인

```
(my_tensorflow) $ pip list | grep horovod
horovod 0.23.0
(my_tensorflow) $ python
>>> import horovod
>>> horovod.__version__
'0.23.0'
```

4. Horovod 실행 예시

1) interactive 실행 예시

```
$ salloc --partition=cas_v100_4 -J debug --nodes=2 --ntasks-per-node=2 --time=08:00:00 --gres=gpu:2 --comment=tensorflow
$ echo $SLURM_NODELIST
gpu[12-13]
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ source activate my_tensorflow
(my_tensorflow) $ horovodrun -np 4 -H gpu12:2,gpu13:2 python tensorflow2_mnist.py
```

2) batch 스크립트 실행 예시

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment tensorflow

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1

source activate my_tensorflow

horovodrun -np 2 python tensorflow2_mnist.py
```

나. Pytorch-horovod 설치

1. 콘다 환경 생성

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1 cmake/3.16.9
$ conda create -n my_pytorch
$ source activate my_pytorch
(my_pytorch) $
```

2. Pytorch 설치 및 horovod 설치

```
(my_pytorch) $ conda install pytorch=1.11.0 python=3.9 torchvision=0.12.0 torchaudio=0.11.0 cudatoolkit=10.2 -c pytorch
(my_pytorch) $ HOROVOD_WITH_MPI=1 HOROVOD_NCCL_LINK=SHARED HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_WITH_PYTORCH=1 \
pip install --no-cache-dir horovod==0.24.0
```

3. Horovod 설치 확인

```
(my_pytorch) $ pip list | grep horovod
horovod 0.24.0

(my_pytorch) $ python
>>> import horovod
>>> horovod.__version__
'0.24.0'
```

4. Horovod 실행 예시

1) interactive 실행 예시

```
$ salloc --partition=cas_v100_4 -J debug --nodes=2 —ntasks-per-node=2 \
--time=08:00:00 --gres=gpu:2 --comment=pytorch
$ echo $SLURM_NODELIST
gpu[22-23]
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ source activate my_pytorch
(my_pytorch) $ horovodrun -np 4 -H gpu22:2,gpu23:2 python pytorch_ex.py
```

2) batch 스크립트 실행 예시

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment pytorch

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1

source activate my_pytorch

horovodrun -np 2 python pytorch_ex.py
```

Ⅲ-8. 딥러닝 프레임워크 병렬화 (Horovod)

가. Tensorflow에서 Horovod 사용법

다중노드에서 멀티 GPU를 활용할 경우 Horovod를 Tensorflow와 연동하여 병렬화가 가능하다. 아래 예시와 같이 Horovod 사용을 위한 코드를 추가해주면 Tensorflow와 연동이 가능하다. Tensorflow 및 Tensorflow에서 활용 가능한 Keras API 모두 Horovod와 연동이 가능하며 우선 Tensorflow에서 Horovod와 연동하는 방법을 소개한다. (예시: MNIST Dataset 및 LeNet-5 CNN 구조)

※ Tensorflow에서 Horovod 활용을 위한 자세한 사용법은 Horovod 공식 가이드 참조(https://github.com/horovod/horovod#usage)

• Tensorflow에서 Horovod 사용을 위한 import 및 메인 함수에서 Horovod 초기화

```
import horovod.tensorflow as hvd
...
hvd.init()
```

※ horovod.tensorflow: Horovod를 Tensorflow와 연동하기 위한 모듈
※ Horovod를 사용하기 위하여 초기화한다.

• 메인 함수에서 Horovod 활용을 위한 Dataset 설정

```
(x_train, y_train), (x_test, y_test) = \
keras.datasets.mnist.load_data('MNIST-data-%d' % hvd.rank())
```

※ 각 작업별로 접근할 dataset을 설정하기 위하여 Horovod rank에 따라 설정 및 생성한다.

• 메인 함수에서 optimizer에 Horovod 관련 설정 및 broadcast, 학습 진행 수 설정

```
opt = tf.train.AdamOptimizer(0.001 * hvd.size())
opt = hvd.DistributedOptimizer(opt)
global_step = tf.train.get_or_create_global_step()
train_op = opt.minimize(loss, global_step=global_step)
hooks = [hvd.BroadcastGlobalVariablesHook(0),
tf.train.StopAtStepHook(last_step=20000 // hvd.size()), ... ]
```

※ Optimizer에 Horovod 관련 설정을 적용하고 각 작업에 broadcast를 활용하여 전달함
※ 각 작업들의 학습 과정 step을 Horovod 작업 수에 따라 설정함

• Horovod의 프로세스 rank에 따라 GPU Device 할당

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
```

※ 각 GPU 별로 하나의 작업을 Horovod의 local rank에 따라 할당함

• Rank 0 작업에 Checkpoint 설정

```
checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None
…
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
hooks=hooks,
config=config) as mon_sess:
```

※ Checkpoint 저장 및 불러오는 작업은 하나의 프로세스에서 수행되어야 하므로 rank 0번에 설정함

나. Keras에서 Horovod 사용법

Tensorflow에서는 Keras API를 활용할 경우에도 Horovod와 연동하여 병렬화가 가능하다. 아래 예시와 같이 Horovod 사용을 위한 코드를 추가해주면 Keras와 연동이 가능하다. (예시: MNIST Dataset 및 LeNet-5 CNN 구조)

※ Keras에서 Horovod 활용을 위한 자세한 사용법은 Horovod 공식 가이드 참조 (https://github.com/horovod/horovod/blob/master/docs/keras.rst)

• Keras에서 Horovod 사용을 위한 import 및 메인 함수에서 Horovod 초기화

```
import horovod.tensorflow.keras as hvd
...
hvd.init()
```

※ horovod.tensorflow.keras: Horovod를 Tensorflow 내의 Keras와 연동하기 위한 모듈

※ Horovod를 사용하기 위하여 초기화한다.

• Horovod의 프로세스 rank에 따라 GPU Device 할당

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
```

※ 각 GPU 별로 하나의 작업을 Horovod의 local rank에 따라 할당함

• 메인 함수에서 optimizer에 Horovod 관련 설정 및 broadcast, 학습 진행 수 설정

```
epochs = int(math.ceil(12.0 / hvd.size()))
...
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
opt = hvd.DistributedOptimizer(opt)
callbacks = [ hvd.callbacks.BroadcastGlobalVariablesCallback(0), ]
```

※ 각 작업들의 학습 과정 step을 Horovod 작업 수에 따라 설정함

※ Optimizer에 Horovod 관련 설정을 적용하고 각 작업에 broadcast를 활용하여 전달함

• Rank 0 작업에 Checkpoint 설정

```
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
```

※ Checkpoint 저장 및 불러오는 작업은 하나의 프로세스에서 수행되어야 하여 rank 0번에 설정함

• Horovod의 프로세스 rank에 따라 GPU Device 할당

```
model.fit(x_train, y_train, batch_size=batch_size, callbacks=callbacks, epochs=epochs,
verbose=1 if hvd.rank() == 0 else 0, validation_data=(x_test, y_test))
```

※ 학습 중 출력되는 문구를 Rank 0번 작업에서만 출력하기 위하여 Rank 0번 작업만 verbose 값을 1로 설정함

다. PyTorch에서 Horovod 사용법

다중노드에서 멀티 GPU를 활용할 경우 Horovod를 PyTorch와 연동하여 병렬화가 가능하다. 아래 예시와 같이 Horovod 사용을 위한 코드를 추가해주면 PyTorch와 연동이 가능하다.

(예시: MNIST Dataset 및 LeNet-5 CNN 구조)

※ PyTorch에서 Horovod 활용을 위한 자세한 사용법은 Horovod 공식 가이드 참조 (https://github.com/horovod/horovod/blob/master/docs/pytorch.rst)

- PyTorch에서 Horovod 사용을 위한 import 및 메인 함수에서 Horovod 초기화 및 설정

```
import torch.utils.data.distributed
import horovod.torch as hvd
...
hvd.init()
if args.cuda:
    torch.cuda.set_device(hvd.local_rank())
    torch.set_num_threads(1)
```

※ torch.utils.data.distributed: PyTorch에서 distributed training을 수행하기 위한 모듈

※ horovod.torch: Horovod를 PyTorch와 연동하기 위한 모듈

※ Horovod 초기화 및 초기화 과정에서 설정된 rank에 따라 작업을 수행할 device를 설정한다.

※ 각 작업별로 CPU thread 1개를 사용하기 위해 torch.set_num_threads(1)를 사용한다.

- Training 과정에 Horovod 관련 내용 추가

```
def train(args, model, device, train_loader, optimizer, epoch):
...
train_sampler.set_epoch(epoch)
...
    if batch_idx % args.log_interval == 0:
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_sampler),
                100.* batch_idx / len(train_loader), loss.item()))
```

※ train_sampler.set_epoch(epoch): train sampler의 epoch 설정

※ Training dataset이 여러 작업들에 나뉘어서 처리되므로 전체 dataset 크기 확인을 위하여 len(train_sampler)을 사용한다.

• Horovod를 활용하여 평균값 계산

```
def metric_average(val, name):
tensor = torch.tensor(val)
avg_tensor = hvd.allreduce(tensor, name=name)
return avg_tensor.item()
```

※ 여러 노드에 걸쳐 평균값을 계산하기 위하여 Horovod의 Allreduce 통신을 활용하여 계산한다.

• Test 과정에 Horovod 관련 내용 추가

```
test_loss /= len(test_sampler)
test_accuracy /= len(test_sampler)
test_loss = metric_average(test_loss, 'avg_loss')
test_accuracy = metric_average(test_accuracy, 'avg_accuracy')
if hvd.rank() == 0:
    print('\nTest set: Average loss: {:.4f}, Accuracy: {:.2f}%\n'.format(
            test_loss, 100. * test_accuracy))
```

※ 여러 노드에 걸쳐 평균값을 계산해야 하므로 위에서 선언된 metric_average 함수를 활용한다.

※ 각 노드별로 Allreduce 통신을 거쳐 loss 및 accuracy에 대해 계산된 값을 동일하게 가지고 있으므로 rank 0번에서 print 함수를 수행한다.

• 메인 함수에서 Horovod 활용을 위한 Dataset 설정

```
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,)) ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
test_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=False, transform=transforms.Compose([
transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,)) ]))
test_sampler = torch.utils.data.distributed.DistributedSampler(
test_dataset, num_replicas=hvd.size(), rank=hvd.rank())
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=args.test_batch_size,
sampler=test_sampler, **kwargs)
```

※ 각 작업별로 접근할 dataset을 설정하기 위하여 Horovod rank에 따라 설정 및 생성한다.

※ PyTorch의 distributed sampler를 설정하여 이를 data loader에 할당한다.

• 메인 함수에서 optimizer에 Horovod 관련 설정 및 training, test 과정에 sampler 추가

```
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(), momentum=args.momentum)
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
optimizer = hvd.DistributedOptimizer(optimizer, named_parameters=model.named_parameters())
for epoch in range(1, args.epochs + 1):
    train(args, model, train_loader, optimizer, epoch, train_sampler)
    test(args, model, test_loader, test_sampler)
```

※ Optimizer에 Horovod 관련 설정을 적용하고 각 작업에 broadcast를 활용하여 전달함

※ Training 및 test 과정에 sampler를 추가하여 각 함수에 전달함

Ⅲ-9. AI 멀티노드 활용

뉴론 시스템에서 딥러닝 학습에 필요한 계산을 멀티 노드를 이용하여 수십 개의 GPU에 나누어 동시에 처리하고, 고속 네트워크를 통한 결과를 합산하는 분산 학습을 위한 여러 방법을 소개한다.

가. HOROVOD

Horovod는 고성능 분산 컴퓨팅 환경에서 노드간 메시지 전달 및 통신관리를 위해 일반적인 표준 MPI 모델을 사용하며, Horovod의 MPI구현은 표준 Tensorflow 분산 훈련 모델보다 간소화된 프로그래밍 모델을 제공한다.

1. HOROVOD (Tensorflow) 설치 및 확인

1) 설치 방법

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1 cmake/3.16.9

$ conda create -n my_tensorflow
$ source activate my_tensorflow

(my_tensorflow) $ conda install tensorflow-gpu=2.0.0 tensorboard=2.0.0 \
 tensorflow-estimator=2.0.0 python=3.7 cudnn cudatoolkit=10 nccl=2.8.3

(my_tensorflow) $ HOROVOD_WITH_MPI=1 HOROVOD_GPU_OPERATIONS=NCCL \
HOROVOD_NCCL_LINK=SHARED HOROVOD_WITH_TENSORFLOW=1 \
pip install --no-cache-dir horovod==0.23.0
```

2) 설치 확인

```
(my_tensorflow) $ pip list | grep horovod
horovod 0.23.0

(my_tensorflow) $ python
>>> import horovod
>>> horovod.__version__ '0.23.0‘

(my_tensorflow) $ horovodrun -cb
Horovod v0.23.0:

 Available Frameworks:
    [X] TensorFlow
    [X] PyTorch
    [ ] MXNet

 Available Controllers:
    [X] MPI
    [X] Gloo

 Available Tensor Operations:
    [X] NCCL
    [ ] DDL
    [ ] CCL
    [X] MPI
    [X] Gloo
```

2. HOROVOD (Tensorflow) 실행 예제

1) 작업제출 스크립트를 이용한 실행

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment tensorflow

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1

source activate my_tensorflow

horovodrun -np 2 python tensorflow2_mnist.py
```

2) 인터렉티브 작업제출을 이용한 실행

```
$ salloc --partition=cas_v100_4 -J debug --nodes=2 -n 4 —time=08:00:00 \
--gres=gpu:2 --comment=tensorflow

$ echo $SLURM_NODELIST
gpu[12-13]

$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ source activate my_tensorflow

(my_tensorflow) $ horovodrun -np 4 -H gpu12:2,gpu13:2 python tensorflow2_mnist.py
```

3. HOROVOD (Pytorch) 설치 및 확인

1) 설치 방법

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1
$ module load python/3.7.1 cmake/3.16.9

$ conda create -n my_pytorch
$ source activate my_pytorch

(my_pytorch) $ conda install pytorch=1.11.0 python=3.9 \
torchvision=0.12.0  torchaudio=0.11.0 cudatoolkit=10.2 -c pytorch
(my_pytorch) $ HOROVOD_WITH_MPI=1 HOROVOD_NCCL_LINK=SHARED \
 HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_WITH_PYTORCH=1 \
pip install --no-cache-dir horovod==0.24.0
```

2) 설치 확인

```
(my_pytorch) $ pip list | grep horovod
horovod 0.24.0

(my_pytorch) $ python
>>> import horovod
>>> horovod.__version__
'0.24.0'

(my_pytorch) $ horovodrun -cb
Horovod v0.24.0:

Available Frameworks:
    [ ] TensorFlow
    [X] PyTorch
    [ ] MXNet

Available Controllers:
    [X] MPI
    [X] Gloo

Available Tensor Operations:
    [X] NCCL
    [ ] DDL
    [ ] CCL
    [X] MPI
    [X] Gloo
```

4. HOROVOD (Pytorch) 실행 예제

1) 작업제출 스크립트 예제

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment pytorch

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1

source activate my_pytorch

horovodrun -np 2 python pytorch_ex.py
```

2) 인터렉티브 작업제출을 이용한 실행

```
$ salloc --partition=cas_v100_4 -J debug --nodes=2 -n 4 —time=08:00:00 \
 --gres=gpu:2 --comment=pytorch

$ echo $SLURM_NODELIST gpu[22-23]

$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ source activate my_pytorch

(my_pytorch) $ horovodrun -np 4 -H gpu22:2,gpu23:2 python pytorch_ex.py
```

나. GLOO

- GLOO는 Facebook에서 개발한 오픈 소스 집단 커뮤니케이션 라이브러리로, horovod에 포함되어 있으며 사용자가 MPI를 설치하지 않고도 horovod를 이용한 멀티노드 작업 수행을 지원한다.
- GLOO의 설치는 horovod에 종속성을 가지며, horovod를 설치하는 과정에서 같이 설치된다.

※ horovod의 설치 방법은 상단의 horovod 설치 및 확인을 참고

1. GLOO 실행 예제

1) 작업제출 스크립트 예제

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment tensorflow

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
source activate my_tensorflow

horovodrun --gloo -np 4 python tensorflow2_mnist.py
```

2) 인터렉티브 작업제출을 이용한 실행

```
$ salloc --partition=cas_v100_4 -J debug --nodes=2 -n 4 —time=08:00:00 \
 --gres=gpu:2 --comment=tensorflow

$ echo $SLURM_NODELIST
  gpu[12-13]

$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ source activate my_tensorflow

(my_tensorflow) $ horovodrun -np 4 --gloo --network-interface ib0 \
-H gpu12:2,gpu13:2 python tensorflow2_mnist.py
```

다. Ray

Ray는 멀티노드 환경에서 병렬 실행을 위한 python 기반의 워크로드를 제공한다. 다양한 라이브러리를 사용하여 pytorch와 같은 딥러닝 모델에서 사용할 수 있다. 자세한 내용은 다음 홈페이지에서 확인할 수 있다.
https://docs.ray.io/en/latest/cluster/index.html

1. Ray 설치 및 단일 노드 실행 예제

1) 설치 방법

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
$ conda create -n my_ray
$ source activate my_ray

(my_ray) $ pip install --no-cache-dir ray

$ export PATH=/home01/${USER}/.local/bin:$PATH
```

2) 작업제출 스크립트 예제

```
#!/bin/bash
#SBATCH -J test_job
#SBATCH -p cas_v100_4
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:2
#SBATCH --comment xxx

module purge
module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1
source activate my_ray

python test.py
```

- test.py

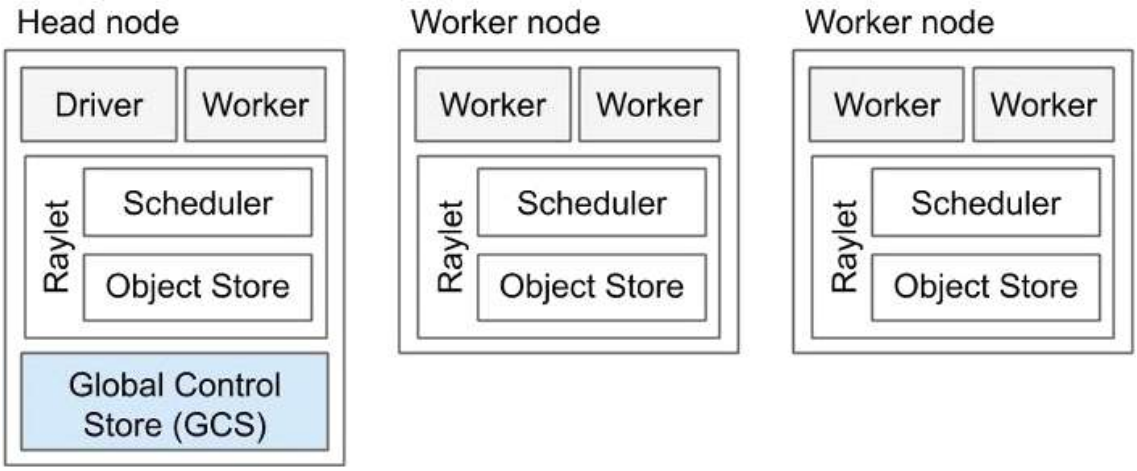
```
import ray

ray.init()

@ray.remote
def f(x):
      return x * x

 futures = [f.remote(i) for i in range(4)]

 print(ray.get(futures)) # [0, 1, 4, 9]

 @ray.remote
 class Counter(object):
      def __init__(self):
          self.n = 0

      def increment(self):
          self.n += 1

      def read(self):
          return self.n

counters = [Counter.remote() for i in range(4)]
[c.increment.remote() for c in counters]
futures = [c.read.remote() for c in counters]
print(ray.get(futures)) # [1, 1, 1, 1]
```

3) 인터렉티브 작업제출을 이용한 실행

```
$ salloc --partition=cas_v100_4 -J debug --nodes=1 -n 4 —time=08:00:00 \
 --gres=gpu:2 --comment=xxx
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1 python/3.7.1


$ source activate my_ray


(my_tensorflow) $ python test.py
[0, 1, 4, 9]
[1, 1, 1, 1]
```

2. Ray Cluster 설치 및 멀티 노드 실행 예제

Ray는 멀티노드 실행 시 하나의 head 노드와 다수의 worker 노드로 실행되며, 태스크를 할당된 자원에 효율적으로 스케줄링해 준다.



https://docs.ray.io/en/latest/cluster/index.html

NERSC에서 작성한 예제는 GITHUB(https://github.com/NERSC/slurm-ray-cluster.git)를 통해 다운로드할 수 있고, 다음과 같이 실행할 수 있다.

1) 설치 방법

```
$ git clone https://github.com/NERSC/slurm-ray-cluster.git
$ cd slurm-ray-cluster

[edit submit-ray-cluster.sbatch, start-head.sh, start-worker.sh]

$ sbatch submit-ray-cluster.sbatch
```

2) 작업제출 스크립트 예제

```
#!/bin/bash

#SBATCH -J ray_test
#SBATCH -p cas_v100_4
#SBATCH --time=00:10:00

### This script works for any number of nodes, Ray will find and manage all resources
#SBATCH --nodes=2

### Give all resources to a single Ray task, ray can manage the resources internally
#SBATCH --ntasks-per-node=4
#SBATCH --comment etc
#SBATCH --gres=gpu:2

# Load modules or your own conda environment here
module load gcc/8.3.0 cuda/10.0 cudampi/openmpi-3.1.0 python/3.7.1
source activate my_ray
export PATH=/home01/${USER} /.local/bin:$PATH


...

#### call your code below
python test.py
exit
```

- start-head.sh

```
#!/bin/bash

#export LC_ALL=C.UTF-8
#export LANG=C.UTF-8

echo "starting ray head node"
# Launch the head node
ray start --head --node-ip-address=$1 --port=12345 --redis-password=$2
sleep infinity
```

- start-worker.sh

```
#!/bin/bash

#export LC_ALL=C.UTF-8
#export LANG=C.UTF-8

echo "starting ray worker node"
ray start --address $1 --redis-password=$2
sleep infinity
```

3) output

```
==========================================
SLURM_JOB_ID = 107575
SLURM_NODELIST = gpu[21-22]
==========================================
        'cuda/10.0' supports the {CUDA_MPI}.
IP Head: 10.151.0.21:12345
STARTING HEAD at gpu21
starting ray head node
2022-04-27 10:57:38,976 INFO services.py:1092 -- View the Ray dashboard at http://localhost:8265
2022-04-27 10:57:38,599 INFO scripts.py:467 -- Local node IP: 10.151.0.21
2022-04-27 10:57:39,000 SUCC scripts.py:497 -- --------------------
2022-04-27 10:57:39,000 SUCC scripts.py:498 -- Ray runtime started.
2022-04-27 10:57:39,000 SUCC scripts.py:499 -- --------------------
2022-04-27 10:57:39,000 INFO scripts.py:501 -- Next steps
2022-04-27 10:57:39,000 INFO scripts.py:503 -- To connect to this Ray runtime from another node, run
2022-04-27 10:57:39,000 INFO scripts.py:507 -- ray start --address='10.151.0.21:12345' --redis-password='90268c54-9df9-4d98-b363-ed6ed4d61a61'
2022-04-27 10:57:39,000 INFO scripts.py:509 -- Alternatively, use the following Python code:
2022-04-27 10:57:39,001 INFO scripts.py:512 -- import ray
2022-04-27 10:57:39,001 INFO scripts.py:519 -- ray.init(address='auto', _redis_password='90268c54-9df9-4d98-b363-ed6ed4d61a61')
2022-04-27 10:57:39,001 INFO scripts.py:522 -- If connection fails, check your firewall settings and network configuration.
2022-04-27 10:57:39,001 INFO scripts.py:526 -- To terminate the Ray runtime, run
2022-04-27 10:57:39,001 INFO scripts.py:527 -- ray stop
STARTING WORKER 1 at gpu22
starting ray worker node
2022-04-27 10:58:08,922 INFO scripts.py:591 -- Local node IP: 10.151.0.22
2022-04-27 10:58:09,069 SUCC scripts.py:606 -- --------------------
2022-04-27 10:58:09,069 SUCC scripts.py:607 -- Ray runtime started.
2022-04-27 10:58:09,069 SUCC scripts.py:608 -- --------------------
2022-04-27 10:58:09,069 INFO scripts.py:610 -- To terminate the Ray runtime, run
2022-04-27 10:58:09,069 INFO scripts.py:611 -- ray stop
2022-04-27 10:58:13,915 INFO services.py:1092 -- View the Ray dashboard at http://127.0.0.1:8265
[0, 1, 4, 9]
[1, 1, 1, 1]
```

3. Ray Cluster (pytorch) 설치 및 멀티 노드 실행 예제

1) 설치 방법

```
$ pip install --user torch torchvision torchaudio tabulate tensorboardX

[edit submit-ray-cluster.sbatch]

$ sbatch submit-ray-cluster.sbatch
```

2) 작업제출 스크립트 예제

```
#!/bin/bash

#SBATCH -J ray_test
#SBATCH -p cas_v100_4
#SBATCH --time=00:10:00

### This script works for any number of nodes, Ray will find and manage all resources
#SBATCH --nodes=2

### Give all resources to a single Ray task, ray can manage the resources internally
#SBATCH --ntasks-per-node=4
#SBATCH --comment etc
#SBATCH --gres=gpu:2

# Load modules or your own conda environment here
module load gcc/8.3.0 cuda/10.0 cudampi/openmpi-3.1.0 python/3.7.1
source activate my_ray
export PATH=/home01/${USER} /.local/bin:$PATH

...

#### call your code below
python examples/mnist_pytorch_trainable.py
exit
```

- mnist_pytorch_trainable.py

```
from __future__ import print_function

import argparse
import os
import torch
import torch.optim as optim

import ray
from ray import tune
from ray.tune.schedulers import ASHAScheduler
from ray.tune.examples.mnist_pytorch import (train, test, get_data_loaders,
                                             ConvNet)

...

 ray.init(address='auto', _node_ip_address=os.environ["ip_head"].split(":")[0], _redis_password=os.environ["redis_password"])
       sched = ASHAScheduler(metric="mean_accuracy", mode="max")
       analysis = tune.run(TrainMNIST,
                           scheduler=sched,
                           stop={"mean_accuracy": 0.99,
                                 "training_iteration": 100},
                           resources_per_trial={"cpu":10, "gpu": 1},
                           num_samples=128,
                           checkpoint_at_end=True,
                           config={"lr": tune.uniform(0.001, 1.0),
                                   "momentum": tune.uniform(0.1, 0.9),
                                   "use_gpu": True})
 print("Best config is:", analysis.get_best_config(metric="mean_accuracy", mode="max"))
```

3) output

```
...


== Status ==
Memory usage on this node: 57.4/377.4 GiB
Using AsyncHyperBand: num_stopped=11
Bracket: Iter 64.000: None | Iter 16.000: 0.9390625000000001 | Iter 4.000: 0.85625 | Iter 1.000: 0.534375
Resources requested: 80/80 CPUs, 8/8 GPUs, 0.0/454.88 GiB heap, 0.0/137.26 GiB objects
Result logdir: /home01/${USER}/ray_results/TrainMNIST_2022-04-27_11-24-20
Number of trials: 20/128 (1 PENDING, 8 RUNNING, 11 TERMINATED)
+-----------------------+------------+-----------------+----------+------------+----------+--------+------------------+
| Trial name            | status     | loc             |       lr | momentum |      acc |   iter |   total time (s) |
|-----------------------+------------+-----------------+----------+------------+----------+--------+------------------|
| TrainMNIST_25518_00000 | RUNNING    | 10.151.0.22:4128 | 0.0831396 |   0.691582 |  0.953125 |     26 |          4.98667 |
| TrainMNIST_25518_00001 | RUNNING    | 10.151.0.22:4127 | 0.0581841 |   0.685648 |  0.94375  |     25 |          5.08798 |
| TrainMNIST_25518_00013 | RUNNING    |                 | 0.0114732 |   0.386122 |           |        |                  |
| TrainMNIST_25518_00014 | RUNNING    |                 | 0.686305  |   0.546706 |           |        |                  |
| TrainMNIST_25518_00015 | RUNNING    |                 | 0.442195  |   0.525069 |           |        |                  |
| TrainMNIST_25518_00016 | RUNNING    |                 | 0.647866  |   0.397167 |           |        |                  |
| TrainMNIST_25518_00017 | RUNNING    |                 | 0.479493  |   0.429876 |           |        |                  |
| TrainMNIST_25518_00018 | RUNNING    |                 | 0.341561  |   0.42485  |           |        |                  |
| TrainMNIST_25518_00019 | PENDING    |                 | 0.205629  |   0.551851 |           |        |                  |
| TrainMNIST_25518_00002 | TERMINATED |                 | 0.740295  |   0.209155 |  0.078125 |      1 |          0.206633 |
| TrainMNIST_25518_00003 | TERMINATED |                 | 0.202496  |   0.102844 |  0.853125 |      4 |          1.17559 |
| TrainMNIST_25518_00004 | TERMINATED |                 | 0.431773  |   0.449912 |  0.85625  |      4 |          0.811173 |
| TrainMNIST_25518_00005 | TERMINATED |                 | 0.595764  |   0.643525 |  0.121875 |      1 |          0.214556 |
| TrainMNIST_25518_00006 | TERMINATED |                 | 0.480667  |   0.412854 |  0.728125 |      4 |          0.885571 |
| TrainMNIST_25518_00007 | TERMINATED |                 | 0.544958  |   0.280743 |  0.15625  |      1 |          0.185517 |
| TrainMNIST_25518_00008 | TERMINATED |                 | 0.277231  |   0.258283 |  0.48125  |      1 |          0.186344 |
| TrainMNIST_25518_00009 | TERMINATED |                 | 0.87852   |   0.28864  |  0.10625  |      1 |          0.203304 |
| TrainMNIST_25518_00010 | TERMINATED |                 | 0.691046  |   0.351471 |  0.103125 |      1 |          0.23274 |
| TrainMNIST_25518_00011 | TERMINATED |                 | 0.926629  |   0.17118  |  0.121875 |      1 |          0.267205 |
| TrainMNIST_25518_00012 | TERMINATED |                 | 0.618234  |   0.881444 |  0.046875 |      1 |          0.226228 |
+-----------------------+------------+-----------------+----------+------------+----------+--------+------------------+

...
```

라. Submit it

Submitit은 Slurm 클러스터 내에서 계산을 위해 Python 함수를 제출하기 위한 경량 도구이다. 기본적으로 스케줄러에 제출된 내용을 정리하여 결과, 로그 등에 대한 액세스를 제공한다.

1. 예제 (1)

- add.py

```
#!/usr/bin/env python3

import submitit

def add(a, b):
    return a + b

# create slurm wrapper object
executor = submitit.AutoExecutor(folder="log_test")

# update slurm parameters
executor.update_parameters(
    partition='cas_V100_2',
    comment='python' )

# submit job
job = executor.submit(add, 5, 7)

# print job ID
print(job.job_id)

# print result
print(job.result())
```

```
$ ./add.py
110651
12
```

- 110651_submission.sh

```
#!/bin/bash

# Parameters
#SBATCH --comment="python"
#SBATCH --error=/scratch/${USER}/2022/02-submitit/test/log_test/%j_0_log.err
#SBATCH --job-name=submitit
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --open-mode=append
#SBATCH --output=/scratch/${USER}/2022/02-submitit/test/log_test/%j_0_log.out
#SBATCH --partition=cas_v100_2
#SBATCH --signal=USR1@90
#SBATCH --wckey=submitit

# command
export SUBMITIT_EXECUTOR=slurm
srun
    --output /scratch/${USER}/2022/02-submitit/test/log_test/%j_%t_log.out
    --error /scratch/${USER}/2022/02-submitit/test/log_test/%j_%t_log.err
    --unbuffered
    /scratch/${USER}/.conda/submitit/bin/python3
    -u
    -m submitit.core._submit
    /scratch/${USER}/2022/02-submitit/test/log_test
```

• Python function stored as pickle file

```
-rw-r--r-- 1 testuser01 tu0000   0 May  4 21:28 log_test/110651_0_log.err
-rw-r--r-- 1 testuser01 tu0000 476 May  4 21:28 log_test/110651_0_log.out
-rw-r--r-- 1 testuser01 tu0000  26 May  4 21:28 log_test/110651_0_result.pkl
-rw-r--r-- 1 testuser01 tu0000 634 May  4 21:28 log_test/110651_submitted.pkl
-rw-r--r-- 1 testuser01 tu0000 735 May  4 21:28 log_test/110651_submission.sh
```

2. 예제 (2)

- add.py

```
#!/usr/bin/env python3

 def add(a, b):
     return a + b

 print(add(5, 7))
```

- run.sh

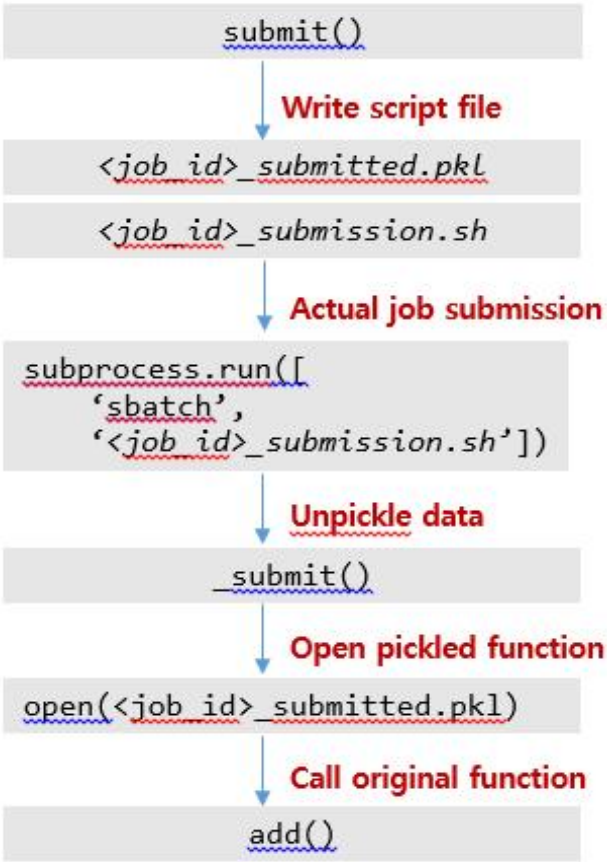
```
#!/bin/bash

#SBATCH --comment="python"
#SBATCH --job-name=submitit
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=cas_v100_2

./add.py
```

```
$ sbatch run.sh
110650
```

- add.py

```
#!/usr/bin/env python3

 import submitit

 def add(a, b):
     return a + b

 # create slurm wrapper object
executor = submitit.AutoExecutor(folder="log_test")

 # update slurm parameters
executor.update_parameters(
     partition='cas_V100_2',
     comment='python'
)

 # submit job
job = executor.submit(add, 5, 7)

 # print job ID
print(job.job_id)

 # print result
print(job.result())
```

```
$ ./add.py
110651
12
```



(Submit multiple jobs in parallel, resource scan (ngpus, ntasks, etc) and capture output for postprocessing)

3. Submit: Multitask Job 예제

- add_para.py

```
#!/usr/bin/env python3

 import submitit

 def add(a, b):
     return a + b

 # create slurm object
executor = submitit.AutoExecutor(folder="log_test")

 # slurm parameters
executor.update_parameters(
     partition='cas_V100_2',
     comment='python',
     ntasks_per_node=3)
)

 # submit job
job = executor.submit(add, 5, 7)

 # print job ID
print(job.job_id)

 # print results
print(job.results())
```

```
$ ./add_para.py
110658
[12, 12, 12]
```

- submitit/submitit/slurm/slurm.py

```
def _make_sbatch_string(
    command: str,
    folder: tp.Union[str, Path],
    job_name: str = "submitit",
    partition: tp.Optional[str] = None,
    time: int = 5,
    nodes: int = 1,
    ntasks_per_node: tp.Optional[int] = None,
    cpus_per_task: tp.Optional[int] = None,
    cpus_per_gpu: tp.Optional[int] = None,
    num_gpus: tp.Optional[int] = None,  # legacy
    gpus_per_node: tp.Optional[int] = None,
    gpus_per_task: tp.Optional[int] = None,
    qos: tp.Optional[str] = None,  # quality of service
    setup: tp.Optional[tp.List[str]] = None,
    mem: tp.Optional[str] = None,
    mem_per_gpu: tp.Optional[str] = None,
    mem_per_cpu: tp.Optional[str] = None,
    signal_delay_s: int = 90,
    comment: tp.Optional[str] = None,
    constraint: tp.Optional[str] = None,
    exclude: tp.Optional[str] = None,
    account: tp.Optional[str] = None,
    gres: tp.Optional[str] = None,
    exclusive: tp.Optional[tp.Union[bool, str]] = None,
    array_parallelism: int = 256,
    wckey: str = "submitit",
    stderr_to_stdout: bool = False,
    map_count: tp.Optional[int] = None,  # used internally
    additional_parameters: tp.Optional[tp.Dict[str, tp.Any]] = None,
    srun_args: tp.Optional[tp.Iterable[str]] = None ) -> str:
```

```
 - 110658_0_log.out
submitit INFO (2022-04-08 12:32:19,583)
- Starting with JobEnvironment(job_id=110658, hostname=gpu25, local_rank=0(3), node=0(1),
global_rank=0(3)) submitit INFO (2022-04-08 12:32:19,584)
- Loading pickle: /scratch/${USER}/2022/02-submitit/test/log_test/110658_submitted.pkl
submitit INFO (2022-04-08 12:32:19,612) - Job completed successfully

 - 110658_1_log.out submitit
INFO (2022-04-08 12:32:19,584)
- Starting with JobEnvironment(job_id=110658, hostname=gpu25, local_rank=1(3), node=0(1), global_rank=1(3))
submitit INFO (2022-04-08 12:32:19,620)
- Loading pickle: /scratch/${USER}/2022/02-submitit/test/log_test/110658_submitted.pkl
submitit INFO (2022-04-08 12:32:19,624) - Job completed successfully

 - 110658_2_log.out
submitit INFO (2022-04-08 12:32:19,583)
- Starting with JobEnvironment(job_id=110658, hostname=gpu25, local_rank=2(3), node=0(1), global_rank=2(3))
submitit INFO (2022-04-08 12:32:19,676)
- Loading pickle: /scratch/${USER}/2022/02-submitit/test/log_test/110658_submitted.pkl
submitit INFO (2022-04-08 12:32:19,681) - Job completed successfully
```

```
import torch
import torch.multiprocessing as mp
import torchvision

 class NeuralNet(self):
    def __init__(self):

        self.layer1 = …
        self.layer2 = …


#Allow passing an NN object to submit()
      def __call__(self):
         job_env = submitit.JobEnvironment() # local rank, global rank
        dataset = torchvision.dataset.MNIST(…)
        loader  =  torch.utils.data.DataLoader(…)

        mp.spawn(…) # data distributed training

mnist = NeuralNet() … job = executor.submit(mnist)
…
job = executor.submit(mnist)
```

마. NCCL

뉴론 시스템에서 NVIDIA GPU에 최적화된 다중 GPU 및 다중 노드 집단 통신 라이브러리인 NCCL의 설치 방법 및 예제 실행 방법을 소개한다.

1. NCCL 설치 및 확인

1) 설치 방법

https://developer.nvidia.com/nccl/nccl-legacy-downloads에서 설치를 원하는 버전 다운로드

```
$ tar Jxvf nccl_2.11.4-1+cuda11.4_x86_64.txz
```

2) 설치 확인

```
$ ls -al nccl_2.11.4-1+cuda11.4_x86_64/
include/  lib/    LICENSE.txt
```

2. NCCL 실행 예제

1) 실행 예제 다운로드

```
$ git clone https://github.com/1duo/nccl-examples
```

2) 실행 예제 컴파일

```
$ module load gcc/10.2.0 cuda/11.4 cudampi/openmpi-4.1.1

$ cd nccl-example
$ mkdir build
$ cd build
$ cmake -DNCCL_INCLUDE_DIR=$NCCL_HOME/include
                -DNCCL_LIBRARY=$NCCL_HOME/lib/libnccl.so ..
$ make
```

3) 실행 결과 확인

```
$ ./run.sh
Example 1: Single Process, Single Thread, Multiple Devices
Success
Example 2: One Device Per Process Or Thread
[MPI Rank 1] Success
[MPI Rank 0] Success
Example 3: Multiple Devices Per Thread
[MPI Rank 1] Success
[MPI Rank 0] Success
```

4) 위의 Example 3을 이용한 2노드 8GPU 실행 예제

4-1) 작업스크립트

```
#!/bin/sh
#SBATCH -J STREAM
#SBATCH --time=10:00:00 # walltime
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:4
#SBATCH -o %N_%j.out
#SBATCH -e %N_%j.err
#SBATCH -p amd_a100_4

srun build/examples/example-3
```

4-2) 예제 코드

```
//
// Example 3: Multiple Devices Per Thread
//

#include
#include "cuda_runtime.h"
#include "nccl.h"
#include "mpi.h"
#include
#include
#include

#define MPICHECK(cmd) do {                          \
  int e = cmd;                                      \
  if( e != MPI_SUCCESS ) {                          \
    printf("Failed: MPI error %s:%d '%d'\n",        \
        __FILE__,__LINE__, e);                      \
    exit(EXIT_FAILURE);                             \
  }                                                 \
} while(0)

#define CUDACHECK(cmd) do {                         \
  cudaError_t e = cmd;                              \
  if( e != cudaSuccess ) {                          \
    printf("Failed: Cuda error %s:%d '%s'\n",       \
        __FILE__,__LINE__,cudaGetErrorString(e));   \
    exit(EXIT_FAILURE);                             \
  }                                                 \
} while(0)


#define NCCLCHECK(cmd) do {                         \
  ncclResult_t r = cmd;                             \
  if (r!= ncclSuccess) {                            \
    printf("Failed, NCCL error %s:%d '%s'\n",       \
        __FILE__,__LINE__,ncclGetErrorString(r));   \
    exit(EXIT_FAILURE);                             \
  }                                                 \
} while(0)

static uint64_t getHostHash(const char *string) {
    // Based on DJB2, result = result * 33 + char
    uint64_t result = 5381;
    for (int c = 0; string[c] != '\0'; c++) {
```

```
        result = ((result << 5) + result) + string[c];
    }
    return result;
}

static void getHostName(char *hostname, int maxlen) {
    gethostname(hostname, maxlen);
    for (int i = 0; i < maxlen; i++) {
        if (hostname[i] == '.') {
            hostname[i] = '\0';
            return;
        }
    }
}

int main(int argc, char *argv[]) {
    //int size = 32 * 1024 * 1024;
    int size = 5;
    int myRank, nRanks, localRank = 0;

    // initializing MPI
    MPICHECK(MPI_Init(&argc, &argv));
    MPICHECK(MPI_Comm_rank(MPI_COMM_WORLD, &myRank));
    MPICHECK(MPI_Comm_size(MPI_COMM_WORLD, &nRanks));

    // calculating localRank which is used in selecting a GPU
    uint64_t hostHashs[nRanks];
    char hostname[1024];
    getHostName(hostname, 1024);
    hostHashs[myRank] = getHostHash(hostname);
    MPICHECK(MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL,
                           hostHashs, sizeof(uint64_t), MPI_BYTE, MPI_COMM_WORLD));

    for (int p = 0; p < nRanks; p++) {
        if (p == myRank) {
            break;
        }

        if (hostHashs[p] == hostHashs[myRank]) {
            localRank++;
        }
    }

    // each process is using four GPUs
    int nDev = 4;
```

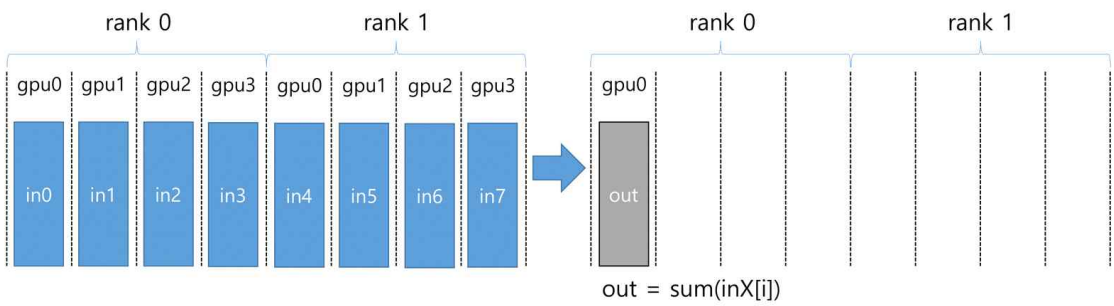
```
int **hostsendbuff = (int **) malloc(nDev * sizeof(int *));
int **hostrecvbuff = (int **) malloc(nDev * sizeof(int *));
int **sendbuff = (int **) malloc(nDev * sizeof(int *));
int **recvbuff = (int **) malloc(nDev * sizeof(int *));

cudaStream_t *s = (cudaStream_t *) malloc(sizeof(cudaStream_t) * nDev);

// picking GPUs based on localRank
for (int i = 0; i < nDev; ++i) {
                hostsendbuff[i] = (int *) malloc(size * sizeof(int));
            for (int j = 0; j < size; j++) {
                hostsendbuff[i][j]=1;
             }
                hostrecvbuff[i] = (int *) malloc(size * sizeof(int));

    CUDACHECK(cudaSetDevice(localRank * nDev + i));
    CUDACHECK(cudaMalloc(sendbuff + i, size * sizeof(int)));
    CUDACHECK(cudaMalloc(recvbuff + i, size * sizeof(int)));
    CUDACHECK(cudaMemcpy(sendbuff[i], hostsendbuff[i], size * sizeof(int),
                         cudaMemcpyHostToDevice));
    CUDACHECK(cudaMemset(recvbuff[i], 0, size * sizeof(int)));
    CUDACHECK(cudaStreamCreate(s + i));
}

ncclUniqueId id;
ncclComm_t comms[nDev];

// generating NCCL unique ID at one process and broadcasting it to all
if (myRank == 0) {
    ncclGetUniqueId(&id);
}

MPICHECK(MPI_Bcast((void *) &id, sizeof(id), MPI_BYTE, 0,
                   MPI_COMM_WORLD));
// initializing NCCL, group API is required around ncclCommInitRank
// as it is called across multiple GPUs in each thread/process
NCCLCHECK(ncclGroupStart());
for (int i = 0; i < nDev; i++) {
    CUDACHECK(cudaSetDevice(localRank * nDev + i));
    NCCLCHECK(ncclCommInitRank(comms + i, nRanks * nDev, id,
                               myRank * nDev + i));
}
NCCLCHECK(ncclGroupEnd());

// calling NCCL communication API. Group API is required when
```

```
    // using multiple devices per thread/process
    NCCLCHECK(ncclGroupStart());
    for (int i = 0; i < nDev; i++) {
NCCLCHECK(ncclAllReduce((const void *) sendbuff[i],
                            (void *) recvbuff[i], size, ncclInt, ncclSum,
                            comms[i], s[i]));
    }
    NCCLCHECK(ncclGroupEnd());

    // synchronizing on CUDA stream to complete NCCL communication
    for (int i = 0; i < nDev; i++) {
        CUDACHECK(cudaStreamSynchronize(s[i]));
    }

    // freeing device memory
    for (int i = 0; i < nDev; i++) {
        CUDACHECK(cudaMemcpy(hostrecvbuff[i], recvbuff[i], size * sizeof(int),
                            cudaMemcpyDeviceToHost));
        CUDACHECK(cudaFree(sendbuff[i]));
        CUDACHECK(cudaFree(recvbuff[i]));

    }

    printf(" \n");
    for (int i = 0; i < nDev; i++) {
        printf("%s rank:%d gpu%d ", hostname, myRank, i);
        for (int j = 0; j < size; j++) {
                printf("%d ", hostsendbuff[i][j]);
        }
        printf("\n");
    }

    printf(" \n");
    for (int i = 0; i < nDev; i++) {
        printf("%s rank:%d gpu%d ", hostname, myRank, i);
        for (int j = 0; j < size; j++) {
                printf("%d ", hostrecvbuff[i][j]);
        }
        printf("\n");
    }

    // finalizing NCCL

    for (int i = 0; i < nDev; i++) {
        ncclCommDestroy(comms[i]);
    }

    // finalizing MPI
```

```
    MPICHECK(MPI_Finalize());
    printf("[MPI Rank %d] Success \n", myRank);
    return 0;
}
```

4-3) 실행 결과

```
gpu37 rank:1 gpu0 1 1 1 1 1 1 1 1
gpu37 rank:1 gpu1 2 2 2 2 2 2 2 2
gpu37 rank:1 gpu2 3 3 3 3 3 3 3 3
gpu37 rank:1 gpu3 4 4 4 4 4 4 4 4

gpu37 rank:1 gpu0 0 0 0 0 0 0 0 0
gpu37 rank:1 gpu1 0 0 0 0 0 0 0 0
gpu37 rank:1 gpu2 0 0 0 0 0 0 0 0
gpu37 rank:1 gpu3 0 0 0 0 0 0 0 0
[MPI Rank 1] Success

gpu36 rank:0 gpu0 1 1 1 1 1 1 1 1
gpu36 rank:0 gpu1 2 2 2 2 2 2 2 2
gpu36 rank:0 gpu2 3 3 3 3 3 3 3 3
gpu36 rank:0 gpu3 4 4 4 4 4 4 4 4

gpu36 rank:0 gpu0 20 20 20 20 20 20 20 20
gpu36 rank:0 gpu1 0 0 0 0 0 0 0 0
gpu36 rank:0 gpu2 0 0 0 0 0 0 0 0
gpu36 rank:0 gpu3 0 0 0 0 0 0 0 0

[MPI Rank 0] Success
```



바. Tensorflow Distribute

Tensorflow Distibute는 멀티 GPU 또는 멀티 서버를 활용하여 분산 훈련을 할 수 있는 Tensorflow API이다. (Tensorflow 2.0 사용)

1. Conda 환경에서 tensorflow 설치 및 확인

1) 설치 방법

```
$ module load python/3.7.1
$ conda create -n tf_test
$ conda activate tf_test
(tf_test) $ conda install tensorflow
```

2) 설치 확인

```
(tf_test) $ conda list | grep tensorflow
tensorflow                2.0.0           gpu_py37h768510d_0
tensorflow-base           2.0.0           gpu_py37h0ec5d1f_0
tensorflow-estimator      2.0.0              pyh2649769_0
tensorflow-gpu            2.0.0                h0d30ee6_0
tensorflow-metadata       1.7.0                   pypi_0    pypi
```

2. 단일 노드, 멀티 GPU 활용(tf.distribute.MirroredStrategy() 사용)

1) 코드 예제(tf_multi_keras.py)

```
import tensorflow as tf
import numpy as np

import os

strategy = tf.distribute.MirroredStrategy()

BUFFER_SIZE = 1000

n_workers = 1
batch_size_per_gpu = 64
global_batch_size = batch_size_per_gpu * n_workers

 def mnist_dataset(batch_size):
        (x_train, y_train), _ = tf.keras.datasets.mnist.load_data()
        x_train = x_train / np.float32(255)
        y_train = y_train.astype(np.int64)
        train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(60000).repeat().batch(batch_size)
        return train_dataset

 def build_and_compile_cnn_model():
        model = tf.keras.Sequential([
                tf.keras.Input(shape=(28, 28)),
                tf.keras.layers.Reshape(target_shape=(28, 28, 1)),
                tf.keras.layers.Conv2D(32, 3, activation='relu'),
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dense(128, activation='relu'),
                tf.keras.layers.Dense(10)
        ])

        model.compile(
                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
                metrics=['accuracy'])
        return model

 dataset = mnist_dataset(global_batch_size)

 with strategy.scope():
        multi_worker_model = build_and_compile_cnn_model()

 multi_worker_model.fit(dataset, epochs=5, steps_per_epoch=BUFFER_SIZE)
```

2) 인터렉티브 작업 제출(1노드 4개 GPU)

```
$ salloc --partition=cas_v100_4 --nodes=1 --ntasks-per-node=4 --gres=gpu:4 --comment=etc
$ conda activate tf_test
(tf_test) $ module load python/3.7.1
(tf_test) $ python tf_multi_keras.py
```

3) 배치 작업 제출 스크립트(1노드 4GPU)(tf_dist_run.sh)

```
#!/bin/bash
#SBATCH -J tf_dist_test
#SBATCH -p cas_v100_4
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -o %x.o%j
#SBATCH -e %s.e%j
#SBATCH --time 00:30:00
#SBATCH --gres=gpu:4
#SBATCH -comment etc

module purge
module load python/3.7.1

source activate tf_test

python tf_multi_keras.py
```

3. 멀티 노드, 멀티 GPU 활용 (tf.distribute.MultiWorkerMirroedStrategy() 사용)

멀티 노드에서 활용하기 위해 전략을 수정하고 각 노드에 환경 변수 TF_CONFIG 설정

1) 코드 예제 수정

```
strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
```

2) 인터렉티브 작업 제출(2노드 각 4 GPU)

```
$ salloc --partition=cas_v100_4 —nodes=2 —ntasks-per-node=4 \
 --gres=gpu:4 --comment=etc
$ sconstrol show hostnames
gpu01
gpu02

gpu01$ conda activate tf_test
(tf_test) $ module load python/3.7.1
(tf_test) $ export TF_CONFIG='{"cluster": {"worker": ["gpu01:12345", "gpu02:12345"]}, "task": {"index": 0, "type": "worker"}}'
(tf_test) $ python tf_multi_keras.py

gpu02$ conda activate tf_test
(tf_test) $ module load python/3.7.1
(tf_test) $ export TF_CONFIG='{"cluster": {"worker": ["gpu01:12345", "gpu02:12345"]}, "task": {"index": 1, "type": "worker"}}'
(tf_test) $ python tf_multi_keras.py
```

3) 배치 작업 제출 스크립트(2노드 각4GPU) (tf_multi_run.sh)

```
#!/bin/bash
#SBATCH -J tf_multi_test
#SBATCH -p 4gpu
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -o ./out/%x.o%j
#SBATCH -e ./out/%x.e%j
#SBATCH --time 01:00:00
###SBATCH --gres=gpu:4
#SBATCH --comment etc

module purge

module load python/3.7.1
source activate tf_test

worker=(`scontrol show hostnames`)
num_worker=${#worker[@]}
PORT=12345

unset TF_CONFIG
for i in ${worker[@]}
do
        tmp_TF_CONFIG=\"$i":"$PORT\"
        TF_CONFIG=$TF_CONFIG", "$tmp_TF_CONFIG
done

TF_CONFIG=${TF_CONFIG:2}
cluster=\"cluster\"": "\{\"worker\"": "\[$TF_CONFIG\]\}

j=0

while [ $j -lt $num_worker ]
do
        task=\"task\"": "\{\"index\"": "$j", "\"type\"": "\"worker\"\}
        tmp=${cluster}", "${task}
        tmp2=\'\{${tmp}\}\'
        ssh ${worker[$j]} "conda activate tf_test; export TF_CONFIG=$tmp2; python tf_multi_keras.py" &
        j=$(($j+1))
done
```

4. 참조

- 케라스(Keras)를 활용한 분산 훈련 (https://www.tensorflow.org/tutorials/distribute/keras)
- [텐서플로2] MNIST 데이터를 훈련 데이터로 사용한 DNN 학습 (http://www.gisdeveloper.co.kr/?p=8534)

사. PytorchDDP

- PytorchDDP(DistributedDataParallel)는 멀티 노드, 멀티 GPU 환경에서 실행할 수 있는 분산 데이터 병렬처리 기능을 제공한다. PytorchDDP 기능 및 튜토리얼은 아래 웹 사이트를 참고 한다.
  - https://pytorch.org/tutorials/intermediate/ddp_tutorial.html
  - https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

아래 예제는 PytorchDDP를 slurm 스케줄러를 통한 실행 방법이다.

1. 작업제출 스크립트 예제

1) 단일노드 예제(단일노드 2GPU)

```
#!/bin/bash -l
#SBATCH -J PytorchDDP
#SBATCH -p cas_v100_4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:2
#SBATCH --comment pytorch
#SBATCH --time 1:00:00
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j

# Configuration
traindata='{path}'
master_port="$((RANDOM%55535+10000))"

# Load software
conda activate pytorchDDP

# Launch one SLURM task, and use torch distributed launch utility
# to spawn training worker processes: one per GPU
srun -N 1 -n 1 python main.py -a config \
                              --dist-url "tcp://127.0.0.1:${master_port}" \
                              --dist-backend 'nccl' \
                              --multiprocessing-distributed \
                              --world-size $SLURM_TASKS_PER_NODE \
                              --rank 0 \
                              $traindata
```

2) 멀티노드 예제(2노드 2GPUs)

```
#!/bin/bash -l
#SBATCH -J PytorchDDP
#SBATCH -p cas_v100_4
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:1
#SBATCH --comment pytorch
#SBATCH --time 10:00:0
#SBATCH -o %x.o%j
#SBATCH -e %x.e%j

# Load software list
module load {module name}
conda activate {conda name}

# Setup node list
nodes=$(scontrol show hostnames $SLURM_JOB_NODELIST) # Getting the node names
nodes_array=( $nodes )
master_node=${nodes_array[0]}
master_addr=$(srun --nodes=1 --ntasks=1 -w $master_node hostname --ip-address)
master_port=$((RANDOM%55535+10000))
worker_num=$(($SLURM_JOB_NUM_NODES))

# Loop over nodes and submit training tasks
for ((  node_rank=0; node_rank<$worker_num; node_rank++ )); do
        node=${nodes_array[$node_rank]}
        echo "Submitting node # $node_rank, $node"
        # Launch one SLURM task per node, and use torch distributed launch utility
        # to spawn training worker processes; one per GPU
        srun -N 1 -n 1 -w $node python main.py -a $config \
                        --dist-url tcp://$master_addr:$master_port \
                        --dist-backend 'nccl' \
                       --multiprocessing-distributed \
                       --world-size $SLURM_JOB_NUM_NODES \
                       --rank $node_rank &

        pids[${node_rank}]=$!
done

# Wait for completion
for pid in ${pids[*]}; do
        wait $pid
done
```