Sbox

Release v1.2

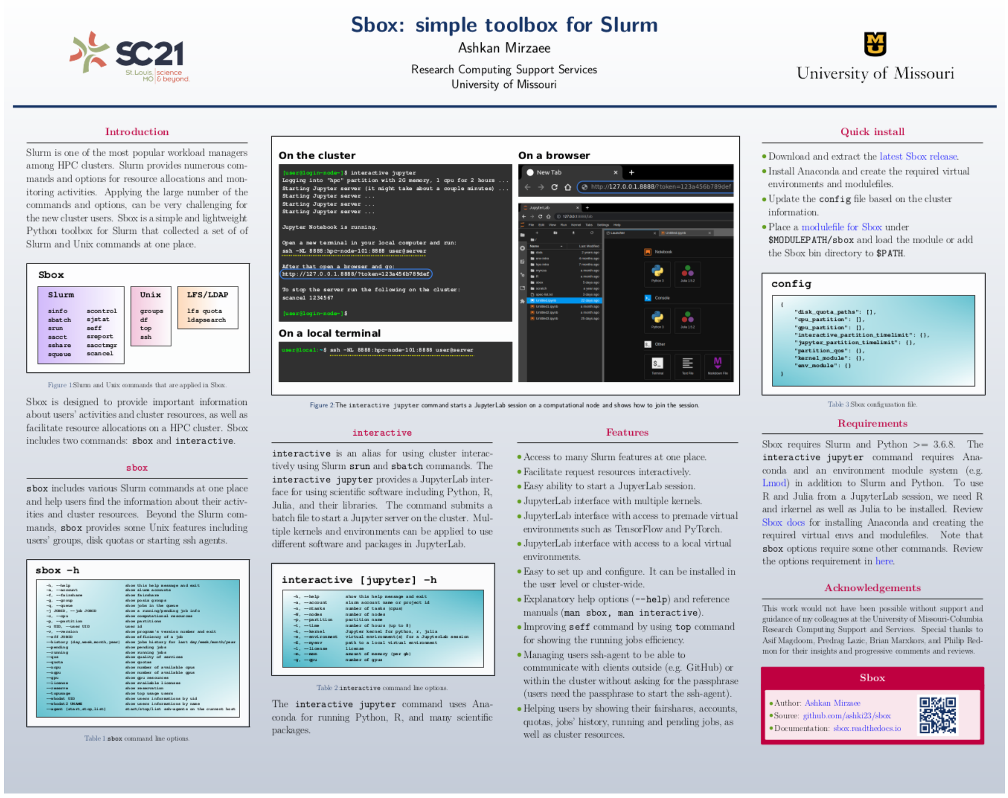
Ashkan Mirzaee

Jul 30, 2022

INSTALL

Sbox is a toolbox for Slurm that provides information about users' accounts and jobs as well as information about the cluster resources. Sbox also can help Slurm admins to collect users' information by user and job IDs. Interactive command uses Slurm `srun` and `sbatch` commands to request resources interactively including running a Jupyter server on the cluster.

Note: The source code and latest release of Sbox is available on the following repository https://github.com/ashki23/sbox.



Sbox: simple toolbox for Slurm

Ashkan Mirzaee
Research Computing Support Services
University of Missouri

SC21
St. Louis, MO | science & beyond
University of Missouri

Introduction

Slurm is one of the most popular workload managers among HPC clusters. Slurm provides numerous commands and options for resource allocations and monitoring activities. Applying the large number of the commands and options, can be very challenging for the new cluster users. Sbox is a simple and lightweight Python toolbox for Slurm that collected a set of of Slurm and Unix commands at one place.

Sbox			
Slurm		Unix	LFS/LDAP
sinfo	scontrol	groups	lfs quota
sbatch	sjstat	df	ldapsearch
srun	seff	top	
sacct	sreport	ssh	
sshare	sacctmgr		
squeue	scancel		

Figure 1:Slurm and Unix commands that are applied in Sbox.

Sbox is designed to provide important information about users' activities and cluster resources, as well as facilitate resource allocations on a HPC cluster. Sbox includes two commands: `sbox` and `interactive`.

sbox

`sbox` includes various Slurm commands at one place and help users find the information about their activities and cluster resources. Beyond the Slurm commands, `sbox` provides some Unix features including users' groups, disk quotas or starting ssh agents.

sbox -h

```
-h, --help                          show this help message and exit
-a, --account                       show slurm accounts
-f, --fairshare                     show fairshare
-g, --group                         show posix groups
-q, --queue                         show jobs in the queue
-j JOBID, --job JOBID               show a running/pending job info
-c, --cpu                           show computational resources
-p, --partition                     show partitions
-u UID, --user UID                  user id
-v, --version                       show program's version number and exit
--eff JOBID                         show efficiency of a job
--history {day,week,month,year}     show jobs history for last day/week/month/year
--pending                           show pending jobs
--running                           show running jobs
--qos                               show quality of services
--quota                             show quotas
--ncpu                              show number of available cpus
--ngpu                              show number of available gpus
--gpu                               show gpu resources
--license                           show available licenses
--reserve                           show reservation
--topusage                          show top usage users
--whodat UID                        show users informations by uid
--whodat2 UNAME                     show users informations by name
--agent {start,stop,list}           start/stop/list ssh-agents on the current host
```

Table 1:`sbox` command line options.

On the cluster

```
[user@login-node-]$ interactive jupyter
Logging into "hpc" partition with 2G memory, 1 cpu for 2 hours ...
Starting Jupyter server (it might take about a couple minutes) ...
Starting Jupyter server ...
Starting Jupyter server ...
Starting Jupyter server ...

Jupyter Notebook is running.

Open a new terminal in your local computer and run:
ssh -NL 8888:hpc-node-101:8888 user@server

After that open a browser and go:
http://127.0.0.1.8888/?token=123a456b789def

To stop the server run the following on the cluster:
scancel 1234567

[user@login-node-]$
```

On a local terminal

```
user@local:~$ ssh -NL 8888:hpc-node-101:8888 user@server
```

On a browser

Figure 2:The `interactive jupyter` command starts a JupyterLab session on a computational node and shows how to join the session.

interactive

`interactive` is an alias for using cluster interactively using Slurm `srun` and `sbatch` commands. The `interactive jupyter` provides a JupyterLab interface for using scientific software including Python, R, Julia, and their libraries. The command submits a batch file to start a Jupyter server on the cluster. Multiple kernels and environments can be applied to use different software and packages in JupyterLab.

interactive [jupyter] -h

```
-h, --help           show this help message and exit
-a, --account        slurm account name or project id
-n, --ntasks         number of tasks (cpus)
-N, --nodes          number of nodes
-p, --partition      partition name
-t, --time           number of hours (up to 8)
-k, --kernel         Jupyter kernel for python, r, julia
-e, --environment    virtual environment(s) for a JupyterLab session
-E, --myenv          path to a local virtual environment
-l, --license        license
-m, --mem            amount of memory (per gb)
-g, --gpu            number of gpus
```

Table 2:`interactive` command line options.

The `interactive jupyter` command uses Anaconda for running Python, R, and many scientific packages.

Features

- Access to many Slurm features at one place.
- Facilitate request resources interactively.
- Easy ability to start a JupyerLab session.
- JupyterLab interface with multiple kernels.
- JupyterLab interface with access to premade virtual environments such as TensorFlow and PyTorch.
- JupyterLab interface with access to a local virtual environments.
- Easy to set up and configure. It can be installed in the user level or cluster-wide.
- Explanatory help options (`--help`) and reference manuals (`man sbox, man interactive`).
- Improving `seff` command by using `top` command for showing the running jobs efficiency.
- Managing users ssh-agent to be able to communicate with clients outside (e.g. GitHub) or within the cluster without asking for the passphrase (users need the passphrase to start the ssh-agent).
- Helping users by showing their fairshares, accounts, quotas, jobs' history, running and pending jobs, as well as cluster resources.

Quick install

- Download and extract the latest Sbox release.
- Install Anaconda and create the required virtual environments and modulefiles.
- Update the `config` file based on the cluster information.
- Place a modulefile for Sbox under `$MODULEPATH/sbox` and load the module or add the Sbox bin directory to `$PATH`.

config

```
{
    "disk_quota_paths": [],
    "cpu_partition": [],
    "gpu_partition": [],
    "interactive_partition_timelimit": {},
    "jupyter_partition_timelimit": {},
    "partition_qos": {},
    "kernel_module": {},
    "env_module": {}
}
```

Table 3:Sbox configuration file.

Requirements

Sbox requires Slurm and Python >= 3.6.8. The `interactive jupyter` command requires Anaconda and an environment module system (e.g. Lmod) in addition to Slurm and Python. To use R and Julia from a JupyterLab session, we need R and irkernel as well as Julia to be installed. Review Sbox docs for installing Anaconda and creating the required virtual envs and modulefiles. Note that `sbox` options require some other commands. Review the options requirement in here.

Acknowledgements

This work would not have been possible without support and guidance of my colleagues at the University of Missouri-Columbia Research Computing Support and Services. Special thanks to Asif Magdoom, Predrag Lazic, Brian Marxkors, and Philip Redmon for their insights and progressive comments and reviews.

Sbox

- Author: Ashkan Mirzaee
- Source: github.com/ashki23/sbox
- Documentation: sbox.readthedocs.io

CHAPTER ONE

FEATURES

- Access to many Slurm features at one place.
- Facilitate request resources interactively.
- Easy ability to start a JupyerLab session.
- JupyterLab interface with multiple kernels.
- JupyterLab interface with access to premade virtual environments such as TensorFlow and PyTorch.
- JupyterLab interface with access to a local virtual environments.
- Easy to set up and configure. It can be installed in the user level or cluster-wide.
- Explanatory help options (`--help`) and reference manuals (`man sbox, man interactive`).
- Improving `seff` command by using `top` command for showing the running jobs efficiency.
- Managing users ssh-agent to be able to communicate with clients outside (e.g. GitHub) or within the cluster (other nodes) without asking for the passphrase.
- Helping users by showing their fairshares, accounts, quotas, jobs' history, running and pending jobs, as well as cluster resources.

1.1 Quick install

- Download and extract the latest Sbox release.
- To access JupyerLab sessions, install Anaconda and create required virtual environments and modulefiles. Review "Requirements" to learn more.
- Update the `config` file based on the cluster information. Review "Configuration" to learn more.
- Place a modulefile for Sbox under `$MODULEPATH/sbox` directory and load the module or add the Sbox bin directory to `$PATH`. A Sbox template modulefile can be found in here.

1.2 Requirements

Sbox requires Slurm and Python >= 3.6.8. The `interactive jupyter` command requires Anaconda and an environment module system (e.g. Lmod) in addition to Slurm and Python. To use R and Julia in JupyterLab sessions, we need R and irkernel as well as Julia to be installed.

Note that Sbox options require some other commands. Review their requirements under the command line options.

The following shows how to install Anaconda and create the required virtual envs and modulefiles.

1.2.1 Python kernel (Anaconda)

The `interactive jupyter` command provides a JupyterLab interface for running Python and many scientific packages by using Anaconda. To install Anaconda, find the latest version of Anaconda from here and run:

```
wget https://repo.anaconda.com/archive/Anaconda3-<year.month>-Linux-x86_64.sh
bash Anaconda3-<year.month>-Linux-x86_64.sh -b -p /<cluster software path>/anaconda/
↪<year.month>
```

In the above lines, update <`year.month`> (e.g. `2021.05`) based on the Anaconda version and <`cluster software path`> (e.g. `/cluster/software/`) based on the cluster path.

To load Anaconda by `modeule load` command, create the following modeulefile under `$MODULEPATH/anaconda/<year.month>.lua`:

```
-- -*- lua -*-

whatis([[Name : anaconda]])
whatis([[Version : <year.month>]])
whatis([[Target : x86_64]])
whatis([[Short description : Python3 distribution including conda and 250+ scientific␣
↪packages.]])
help([[Python3 distribution including conda and 250+ scientific packages.]])

-- Create environment variables
local this_root = "/<cluster software path>/anaconda/<year.month>"

prepend_path("PATH", this_root .. "/bin", ":")
prepend_path("LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("LD_LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("MANPATH", this_root .. "/share/man", ":")
prepend_path("INCLUDE", this_root .. "/include", ":")
prepend_path("C_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("CPLUS_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("PKG_CONFIG_PATH", this_root .. "/lib/pkgconfig", ":")
setenv("ANACONDA_ROOT", this_root)
```

Or adding the following tcl modulefile under `$MODULEPATH/anaconda/<year.month>`:

```
#%Module1.0
## Metadata ###########################################
set this_module        anaconda
set this_version       <year.month>
set this_root          /<cluster software path>/${this_module}/${this_version}
```

(continues on next page)

(continued from previous page)

```
set this_docs          https://docs.anaconda.com/
set this_module_upper  [string toupper ${this_module}]
## Module ###############################################
proc ModulesHelp { } {
        global this_module this_version this_root this_docs
        puts stderr "*****************************************************"
        puts stderr "Name:          ${this_module}"
        puts stderr "Version:       ${this_version}"
        puts stderr "Documentation: ${this_docs}"
        puts stderr "*****************************************************\n"
}

module-whatis "Set up environment for ${this_module} ${this_version}"

prepend-path  PATH  ${this_root}/bin
prepend-path  LIBRARY_PATH  ${this_root}/lib
prepend-path  LD_LIBRARY_PATH  ${this_root}/lib
prepend-path  MANPATH  ${this_root}/share/man
prepend-path  INCLUDE  ${this_root}/include
prepend-path  C_INCLUDE_PATH  ${this_root}/include
prepend-path  CPLUS_INCLUDE_PATH  ${this_root}/include
prepend-path  PKG_CONFIG_PATH  ${this_root}/lib/pkgconfig
setenv        ${this_module_upper}_ROOT  ${this_root}
```

1.2.2 R kernel

Users can run R scripts within a JupterLab notebook by `interactive jupyter -k r`. To have R, irkernel and many other R packages, we can create the following env including r-essentials from Anaconda:

```
cd /<cluster software path>/anaconda/<year.month>
./bin/conda create -n r-essentials-<R version> -c conda-forge r-essentials r-base r-
↪irkernel jupyterlab
```

In the above lines, `<cluster software path>` and `<year.month>` should be updated based on the Anaconda path and version, and `<R version>` (e.g. `4.0.3`) based on the version of R in the env.

The following modulefile should be added to `$MODULEPATH/r-essentials/<R version>.lua` to be able to load the R env:

```
-- -*- lua -*-

whatis([[Name : r-essentials]])
whatis([[Version : <R version>]])
whatis([[Target : x86_64]])
whatis([[Short description : A conda environment for R and 80+ scientific packages.]])
help([[A conda environment for R and 80+ scientific packages.]])

-- Create environment variables
local this_root = "/<cluster software path>/anaconda/envs/r-essentials-<R version>"

prepend_path("PATH", this_root .. "/bin", ":")
```

(continues on next page)

(continued from previous page)

```
prepend_path("LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("LD_LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("MANPATH", this_root .. "/share/man", ":")
prepend_path("INCLUDE", this_root .. "/include", ":")
prepend_path("C_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("CPLUS_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("PKG_CONFIG_PATH", this_root .. "/lib/pkgconfig", ":")
setenv("ANACONDA_ROOT", this_root)
```

Or adding a tcl modulefile similar to the above tcl template for Anaconda.

1.2.3 Julia kernel

The `interactive jupyter -k julia` command provides Julia from a JupyterLab notebook. Julia can be installed from Spack, source or Anaconda. The following shows how to install Julia from Anaconda (Note that if Julia have been installed on the cluster, you can skip this section and use the available Julia module instead).

```
cd /<cluster software path>/anaconda/<year.month>
./bin/conda create -n julia-<version> -c conda-forge julia
```

In the above lines, `<cluster software path>` and `<year.month>` should be updated based on the Anaconda path and version, and `<version>` (e.g. `1.6.1`) based on the version of Julia in the env.

The following modulefile should be added to `$MODULEPATH/julia/<version>.lua`:

```
-- -*- lua -*-

whatis([[Name : julia]])
whatis([[Version : <version>]])
whatis([[Target : x86_64]])
whatis([[Short description : The Julia Language: A fresh approach to technical␣
↪computing]])
help([[The Julia Language: A fresh approach to technical computing]])

-- Create environment variables
local this_root = "/<cluster software path>/anaconda/envs/julia-<version>"

prepend_path("PATH", this_root .. "/bin", ":")
prepend_path("LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("LD_LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("MANPATH", this_root .. "/share/man", ":")
prepend_path("INCLUDE", this_root .. "/include", ":")
prepend_path("C_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("CPLUS_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("PKG_CONFIG_PATH", this_root .. "/lib/pkgconfig", ":")
setenv("ANACONDA_ROOT", this_root)
```

Or adding a tcl modulefile similar to the above tcl template for Anaconda.

Note that the first time that users run `interactive jupyter -k julia`, Julia Jupyter kernal (IJulia) will be installed under `~/.julia`.

1.2.4 On demand Python and R pakages

Popular Python pakages that are not available in Anaconda can be added to `interactive jupyter -e`. For instance the following shows how to create a TensorFlow (TF) env:

```
cd /<cluster software path>/anaconda/<year.month>
./bin/conda create -n tensorflow-gpu-<version> anaconda
./bin/conda install -n tensorflow-gpu-<version> tensorflow-gpu gpustat
```

Similarly, we can create a PyTorch (PT) env with:

```
cd /<cluster software path>/anaconda/<year.month>
./bin/conda create -n pytorch-<version> anaconda
./bin/conda install -n pytorch-<version> -c pytorch pytorch gpustat
```

For instance, we can collect popular R bio packages in the following env from bioconda channel:

```
cd /<cluster software path>/anaconda/<year.month>
./bin/conda create -n r-bioessentials-<version> -c bioconda -c conda-forge bioconductor-
↪edger bioconductor-oligo r-monocle3 r-signac r-seurat scanpy macs2 jupyterlab r-
↪irkernel
```

In the above lines, `<cluster software path>` and `<year.month>` should be updated based on the Anaconda path and version, and `<version>` (e.g. `2.4.1`) based on the version of TF, PT, or R.

For each env, we need to add a modulefile to `$MODULEPATH/<env name>/<version>.lua`. For instance `$MODULEPATH/tensorflow/<version>.lua` is:

```
-- -*- lua -*-

whatis([[Name : tensorflow]])
whatis([[Version : <version>]])
whatis([[Target : x86_64]])
whatis([[Short description : Python3 distribution including TensorFlow and 250+␣
↪scientific packages.]])
help([[Python3 distribution including TensorFlow and 250+ scientific packages.]])

-- Create environment variables
local this_root = "/<cluster software path>/anaconda/envs/tensorflow-gpu-<version>"

prepend_path("PATH", this_root .. "/bin", ":")
prepend_path("LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("LD_LIBRARY_PATH", this_root .. "/lib", ":")
prepend_path("MANPATH", this_root .. "/share/man", ":")
prepend_path("INCLUDE", this_root .. "/include", ":")
prepend_path("C_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("CPLUS_INCLUDE_PATH", this_root .. "/include", ":")
prepend_path("PKG_CONFIG_PATH", this_root .. "/lib/pkgconfig", ":")
setenv("ANACONDA_ROOT", this_root)
```

Or adding a tcl modulefile similar to the above tcl template for Anaconda.

Note: Users can add other packages and mix a local stack of packages with the premade environments. For Python and R packages users can apply `pip install` and `install.packages` respectively to install packages on their home. In order to install packages in a differnt path than home, we can specify the desired path and add the new path to the library path of the software. See examples under the `interactive` command line options examples.

1.3 Configuration

The sbox and interactive commands are reading the required information from the below JSON config file.

```
{
    "disk_quota_paths": [],
    "cpu_partition": [],
    "gpu_partition": [],
    "interactive_partition_timelimit": {},
    "jupyter_partition_timelimit": {},
    "partition_qos": {},
    "kernel_module": {},
    "env_module": {}
}
```

The config file includes:

- disk_quota_paths: A list of paths to the disk for finding users quotas. By default the first input is considered as the users' home path.
- cpu_partition: A list of computational partitions.
- gpu_partition: A list of GPU partitions.
- interactive_partition_timelimit: A dictionary of interactive partitions (i.e. users should access by srun) and their time limits (hour). The first input is considered as the default partition.
- jupyter_partition_timelimit: A dictionary of computational/gpu partitions that users can run Jupter servers interactively and their time limits (hour). The first input is considered as the default partition.
- partition_qos: A dictionary of partitions and the corresponding quality of services.
- kernel_module: A dictionary of kernels and the corresponding modules. A Python kernel is required (review the Requirments).
- env_module: A dictionary of virtual environments and the corresponding modules.

For example:

```
{
    "disk_quota_paths": ["/home", "/data", "/gprs", "/storage/htc"],
    "cpu_partition": ["Interactive","Lewis","Serial","Dtn","hpc3","hpc4","hpc4rc","hpc5",
↪"hpc6","General","Gpu"],
    "gpu_partition": ["Gpu","gpu3","gpu4"],
    "interactive_partition_timelimit": {
        "Interactive": 4,
        "Dtn": 4,
        "Gpu": 2
    },
    "jupyter_partition_timelimit": {
        "Lewis": 8,
        "hpc4": 8,
        "hpc5": 8,
        "hpc6": 8,
        "gpu3": 8,
        "gpu4": 8,
        "Gpu": 2
```

(continues on next page)

(continued from previous page)

```
    },
    "partition_qos": {
        "Interactive": "interactive",
        "Serial": "seriallong",
        "Dtn": "dtn"
    },
    "kernel_module": {
        "python": "anaconda",
        "r": "r-essentials",
        "julia": "julia"
    },
    "env_module": {
        "tensorflow-v1.9": "tensorflow/1.9.0",
        "tensorflow": "tensorflow",
        "pytorch": "pytorch",
        "r-bio": "r-bioessentials"
    }
}
```

1.4 Sbox

`sbox` command includes various Slurm commands at one place. Users can use different options to find the information about the cluster and their accounts and activities. Beyond the Slurm commands, `sbox` provides some Unix features including users' groups, disk quotas and starting ssh agents. The ssh-agent lets users communicate with clients outside the cluster such as GitHub and GitLab or with other nodes within the cluster via ssh without asking for the passphrase (you need the passphrase to start the ssh-agent).

1.4.1 Command line options

- `-h, --help`: Show the help message and exit.
- `-a, --account`: Return user's Slurm accounts by using Slurm `sacctmgr`. If the cluster does not use Slurm for users' account management, it returns empty output.
- `-f, --fairshare`: Return users' fairshare by using Slurm `sshare` command. If the cluster does not follow a fairshare model, it returns empty output.
- `-g, --group`: Return user's posix groups by using Unix `groups` command.
- `-q, --queue`: Return user's jobs in the Slurm queue by Slurm using `squeue` command.
- `-j, --job`: Show a running/pending job info by using Slurm `scontrol` command. It requires a valid job ID as argument.
- `-c, --cpu`: Return computational resources including number of cores and amount of memory on each node. It uses Slurm `sjstat` command.
- `-p, --partition`: Show cluster partitions by using Slurm `sinfo` command.
- `-u, --user`: Store a user ID. By default it uses `$USER` as user ID for any query that needs a user ID. It can be used with other options to find the information for other users.
- `-v, --version`: Show program's version number and exit.

- `--eff`: Show efficiency of a job. It requires a valid job ID as argument. It uses Slurm `seff` command for completed/finished jobs and Unix `top` command for a running job.
- `--history`: Return jobs history for last day, week, month or year. It requires one of the day/week/month/year options as an argument. It uses Slurm `sacct` command and returns empty output if the cluster does not use Slurm for users' account management.
- `--pending`: Return user's pending jobs by using Slurm `squeue` command.
- `--running`: Return user's running jobs by using Slurm `squeue` command.
- `--cancel`: Cancel jobs by a single ID or a comma separated list of IDs using Slurm `scancel` command.
- `--qos`: Show user's quality of services (QOS) and a list of available QOS in the cluster. It uses Slurm `sacctmgr show assoc` command and returns empty output if the cluster does not use Slurm for users' account management.
- `--quota`: Return user's disk quotas. It uses `lfs quota` command for LFS systems and Unix `df` command for NFS systems. It returns pooled size of the disk if the cluster does not have user/group storage accounts.
- `--ncpu`: Show number of available cpus on the cluster using Slurm `sinfo` command.
- `--ncgu`: Show number of available gpus on the cluster using Slurm `squeue` and `sinfo` commands.
- `--gpu`: Show gpu resources including gpu cards' name and numbers using Slurm `sinfo` command.
- `--license`: Show available license servers using Slurm `scontrol` command.
- `--reserve`: Show Slurm reservations using Slurm `scontrol` command.
- `--topusage`: Show top usage users using Slurm `sreport` command.
- `--whodat`: Show users informations by UID. It uses `ldapsearch` command and returns empty output if the cluster does not use LDAP.
- `--whodat2`: Show users informations by name. It uses `ldapsearch`command and returns empty output if the cluster does not use LDAP.
- `--agent`: Start, stop and list user's ssh-agents on the current host. It requires one of the start/stop/list options as an argument. Use `ssh -o StrictHostKeyChecking=no` to disable asking for host key acceptances.
- `--report`: Show current cluster utilization based on the running jobs. It uses Slurm `sinfo` and `squeue` commands.

Examples

Jobs histoty:

```
[user@lewis4-r630-login-node675 ~]$ sbox --hist day
-------------------------------------------------------------------------------- Jobs␣
↪History - Last Day ------------------------------------------------------------------
↪-
    JobID   User Account      State Partition     QOS NCPU NNod ReqMem              ␣
↪Submit   Reserved              Start    Elapsed                 End              ␣
↪NodeList
---------- ------ -------- ---------- --------- -------- ---- ---- ------ -----------------
↪-- ---------- ------------------- ---------- ------------------- --------------------
  23126125  user  general  COMPLETED Interact+ intera+    1    1    2Gn 2021-07-
↪28T01:25:05   00:00:00 2021-07-28T01:25:05   00:00:03 2021-07-28T01:25:08 lewis4-c8k-
↪hpc2-nod+
  23126126  user  general  COMPLETED Interact+ intera+    1    1    2Gn 2021-07-
↪28T01:25:13   00:00:00 2021-07-28T01:25:13   00:00:03 2021-07-28T01:25:16 lewis4-c8k-
↪hpc2-nod+
```

(continues on next page)

(continued from previous page)

```
 23126127  user   general  COMPLETED Interact+ intera+    1    1    2Gn 2021-07-
↪28T01:25:20   00:00:00 2021-07-28T01:25:20   00:00:08 2021-07-28T01:25:28 lewis4-c8k-
↪hpc2-nod+
 23126128  user   genera+  COMPLETED Interact+ intera+    1    1    2Gn 2021-07-
↪28T01:25:49   00:00:00 2021-07-28T01:25:49   00:00:03 2021-07-28T01:25:52 lewis4-c8k-
↪hpc2-nod+
 23126129  user   genera+  COMPLETED Interact+ intera+    1    1    2Gn 2021-07-
↪28T01:26:05   00:00:00 2021-07-28T01:26:05   00:00:06 2021-07-28T01:26:11 lewis4-c8k-
↪hpc2-nod+
 23126130  user   genera+  COMPLETED       Gpu  normal    1    1    2Gn 2021-07-
↪28T01:26:38   00:00:02 2021-07-28T01:26:40   00:00:11 2021-07-28T01:26:51 lewis4-z10pg-
↪gpu3-n+
 23126131  user   genera+ CANCELLED+       Gpu  normal    1    1    2Gn 2021-07-
↪28T01:27:43   00:00:01 2021-07-28T01:27:44   00:01:03 2021-07-28T01:28:47 lewis4-z10pg-
↪gpu3-n+
```

Jobs efficiency for running and compeleted jobs:

```
[user@lewis4-r630-login-node675 ~]$ sbox --eff 23227816
------------------------------------- Job Efficiency -------------------------------------
↪-
   PID USER      PR  NI    VIRT    RES     SHR  S   %CPU   %MEM   TIME+   COMMAND
 47262 user      20   0  115700   3888    1600  S   0.0    0.0    0:00.03 bash
 47346 user      20   0  113292   149298  1256  S   99.0   23.0   0:13.30 python

RES: shows resident memory which is accurate representation of how much actual physical␣
↪memory a process is consuming
%CPU: shows the percentage of the CPU that is being used by the process
```

```
[user@lewis4-r630-login-node675 ~]$ sbox --eff 23126131
------------------------------------- Job Efficiency -------------------------------------
↪-
Job ID: 23126131
Cluster: lewis4
User/Group: user/user
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:11:01
CPU Efficiency: 48.59% of 00:21:03 core-walltime
Memory Utilized: 445.80 MB
Memory Efficiency: 24.24% of 2.00 GB
```

Accounts, fairshares, and groups:

```
[user@lewis4-r630-login-node675 ~]$ sbox -afg
---------------------------------------- Accounts ----------------------------------------
↪-
rcss-gpu  root  general-gpu  rcss  general

--------------------------------------- Fairshare ----------------------------------------
↪-
             Account       User  RawShares  NormShares    RawUsage  EffectvUsage ␣
```

(continues on next page)

(continued from previous page)

```
↪FairShare
-------------------- ---------- ---------- ----------- ----------- ------------- --------
↪--
root                       user     parent    1.000000           0       0.000000    1.
↪000000
general-gpu                user          1    0.000005        3942       0.000016    0.
↪098089
rcss                       user          1    0.001391        1327       0.001147    0.
↪564645
general                    user          1    0.000096     3196356       0.000243    0.
↪174309
rcss-gpu                   user          1    0.000181           0       0.000000    0.
↪999976

----------------------------------------- Groups -----------------------------------------
↪-
user : user rcss gaussian biocompute rcsslab-group rcss-maintenance rcss-cie software-
↪cache
```

Disk quotas:

```
[user@lewis4-r630-login-node675 ~]$ sbox --quo
------------------------------------- user /home storage --------------------------------
↪-----
      File          Used  Use%  Avail  Size  Type
      /home/user    996M  20%   4.1G   5.0G  nfs4
-----------------------------------------------------------------------------------------
↪------
------------------------------------- user /data storage --------------------------------
↪-----
     Filesystem     used   quota   limit   grace   files   quota   limit   grace
          /data  85.89G      0k    105G       - 1477223       0       0       -
-----------------------------------------------------------------------------------------
↪------
```

Jobs in the queue:

```
[user@lewis4-r630-login-node675 ~]$ sbox -q
----------------------------------- Jobs in the Queue -----------------------------------
↪-
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          23150514     Lewis jupyter-    user  R       5:29      1 lewis4-r630-hpc4-
↪node537
```

Cluster resources:

```
[user@lewis4-r630-login-node675 ~]$ sbox --ngpu
------------------------------------- Number of GPUs ------------------------------------
↪-
Partition Gpu has 19 gpus available out of 27 (70%)
Partition gpu3 has 15 gpus available out of 15 (100%)
Partition gpu4 has 4 gpus available out of 12 (33%)
```

```
[user@lewis4-r630-login-node675 ~]$ sbox --ncpu
------------------------------------- Number of CPUs ------------------------------------
↪-
Partition Interactive has 158 cpus available out of 160 (99%)
Partition Lewis has 161 cpus available out of 2344 (7%)
Partition Serial has 42 cpus available out of 48 (88%)
Partition Dtn has 35 cpus available out of 36 (97%)
Partition hpc3 has 24 cpus available out of 456 (5%)
Partition hpc4 has 79 cpus available out of 1008 (8%)
Partition hpc4rc has 58 cpus available out of 952 (6%)
Partition hpc5 has 70 cpus available out of 1400 (5%)
Partition hpc6 has 0 cpus available out of 2976 (0%)
Partition General has 1837 cpus available out of 7008 (26%)
Partition Gpu has 383 cpus available out of 412 (93%)
```

1.5 Interactive

`interactive` is an alias for using cluster interactively using Slurm `srun` and `sbatch` commands. The `interactive jupyter` provides a JupyterLab interface for using scientific software including Python, R, Julia, and their libraries. The command submits a batch file by `sbatch` command and runs a Jupyter server on the cluster. Multiple kernels and environments can be applied to use different software and packages in JupyterLab.

1.5.1 Command line options

- `-h, --help`: Show this help message and exit.
- `-a, --account`: Slurm account name or project ID.
- `-n, --ntasks`: Number of tasks (cpus).
- `-N, --nodes`: Number of nodes.
- `-p, --partition`: Partition name.
- `-t, --time`: Number of hours based on the partitions timelimit.
- `-l, --license`: Add a license to an interactive session.
- `-m, --mem`: Amount of memory (per GB).
- `-g, --gpu`: Number of gpus.
- `-k, --kernel`: Jupyter kernel for python, r, julia. The default kernel is python.
- `-e, --environment`: Virtual environment(s) for a JupyterLab session.
- `-E, --myenv`: Path to a local virtual environment. The local virtual envs should contain JupyterLab.

Examples

Using the cluster interactively:

```
[user@lewis4-r630-login-node675 ~]$ interactive
Logging into Interactive partition with 2G memory, 1 cpu for 2 hours ...
[user@lewis4-r7425-htc5-node835 ~]$
```

Using the cluster interactively with more time and resources:

```
[user@lewis4-r630-login-node675 ~]$ interactive --mem 16 -n 6 -t 4
Logging into Interactive partition with 16G memory, 6 cpu for 4 hours ...
[user@lewis4-r7425-htc5-node835 ~]$
```

Using the cluster interactively with a license:

```
[user@lewis4-r630-login-node675 ~]$ interactive --mem 16 -n 6 -t 4 -l matlab
Logging into Interactive partition with 16G memory, 6 cpu for 4 hours with a matlab␣
↪license ...
[user@lewis4-r7425-htc5-node835 ~]$
```

Using a Gpu interactively:

```
[user@lewis4-r630-login-node675 ~]$ interactive -p Gpu
Logging into Gpu partition with 1 gpu, 2G memory, 1 cpu for 2 hours ...
[user@lewis4-r730-gpu3-node431 ~]$
```

Using JupyterLab:

```
[user@lewis4-r630-login-node675 ~]$ interactive jupyter
Logging into Lewis partition with 2G memory, 1 cpu for 2 hours ...
Starting Jupyter server (it might take about a couple minutes) ...
Starting Jupyter server ...
Starting Jupyter server ...

Jupyter Notebook is running.

Open a new terminal in your local computer and run:
ssh -NL 8888:lewis4-r630-hpc4-node303:8888 user@lewis.rnet.missouri.edu

After that open a browser and go:
http://127.0.0.1:8888/?token=9e223bd179d228e0e334f8f4a85dfd904eebd0ab9ded7e55

To stop the server run the following on the cluster:
scancel 23150533
```

Using JupyterLab with R kernel:

```
[user@lewis4-r630-login-node675 ~]$ interactive jupyter -k r
Logging into Lewis partition with 2G memory, 1 cpu for 2 hours ...
Starting Jupyter server (it might take about a couple minutes) ...
Starting Jupyter server ...
Starting Jupyter server ...
...
```

Using TensorFlow on JupyterLab by a different account and on a partition with 16 GB memory for 8 hours:

```
[user@lewis4-r630-login-node675 ~]$ interactive jupyter -a general-gpu -p gpu3 --mem 16 -
↪t 8 -e tensorflow
Logging into gpu3 partition with 1 gpu, 16G memory, 1 cpu for 8 hours with account␣
↪general-gpu ...
Starting Jupyter server (it might take about a couple minutes) ...
Starting Jupyter server ...
```

(continues on next page)

(continued from previous page)

```
Starting Jupyter server ...
...
```

Note: Users can install other packages and mix local packages with the premade environments. For example, for Python:

```
pip install --target </path/my-packages/lib/> <pkg-name>
export PYTHONPATH=</path/my-packages/lib/>:$PYTHONPATH
```

For R, run the following in R:

```
dir.create("<your/path/for/R/version>")
install.packages("<pkg-name>", repos = "http://cran.us.r-project.org", lib = "<your/path/
↪for/R/version>")
.libPaths("<your/path/for/R/version>")
```

Using a local virtual environment:

```
[user@lewis4-r630-login-node675 ~]$ interactive jupyter -E </path/to/local/env>
Logging into Lewis partition with 2G memory, 1 cpu for 2 hours ...
Starting Jupyter server (it might take about a couple minutes) ...
Starting Jupyter server ...
```

Note: The local environments must include `jupyterlab`. For R environments, they must also contain `r-irkernel`. For instance:

```
conda create -p </path/to/local/env> -c conda-forge r-base jupyterlab r-irkernel
```