BIU Slurm System

Slurm is a Linux open-source resource manager and job scheduler designed to utilize a fair share of the computing resources.
What Are Slurm Partitions?
Partitions in Slurm are sets of compute nodes grouped for specific types of jobs. Each partition has its own limits such as maximum job runtime, hardware (CPUs, GPUs), and access permissions. Partitions are similar to queues in other workload managers.
Programs (jobs) run via Slurm are sent to one or more of the physical servers (Nodes). Slurm is software that helps defining and executing these jobs, as well as managing users, permissions, and resource allocation. It helps track and display job details as well.
Users must select a partition that matches the resources their job needs. If not specified, the default partition will be used.
Slurm login servers – the servers you connect to in order to submit your batch jobs.
Compute nodes – the servers within the partitions where your jobs actually run.

1. BIU Cluster partitions Summary

Cluster General Partitions Summary

Partition Name	Purpose	Nodes	Max Time	Access (Accounts)	MaxJobs Per User	Max GPU /CPU Per Job	Node Mem
generic	General GPU jobs	dsicsgpu[02-09], dsisarit[02,05]	4h	All users	8 jobs	8 GPUs	192G
generic-48G	General GPU jobs	dsiaw15	4h	All users	2 jobs	2 GPUs	128G
H200-4h	GPU jobs on H200	hpc8h200-01	4h	All users	4 jobs	2 GPUs	2T
H200-12h	GPU jobs on H200	hpc8h200-01	12h	All users	4 jobs	2 GPUs	2T
L4-4h	GPU jobs on L4	hpc8l4-01-01	4h	All users	4 jobs	2 GPUs	512G
L4-12h	GPU jobs on L4	hpc8l4-01-01	12h	All users	4 jobs	2 GPUs	512G
A100-4h	GPU jobs on A100	hpc2a100-01	4h	All users	4 jobs	2 GPUs	512G
cpu1T-24h	CPU jobs	hpccpu01	24h	All Users	8 jobs	48 CPUs	1T
cpu192G-48h	CPU jobs	dml[02-25]	48h	All Users	16 jobs	48 CPUs	192G

Cluster Private Partitions Summary

Partition Name	Purpose	Nodes	Max Time	Access (Accounts)	MaxJobs Per User	Max GPU /CPU Per Job	Node Mem
p_kugler	GPU jobs on A100 Kugler group	hpc2a100-01	Unlimited	Users in Prof. Kugler's group	Unlimited	Unlimited	512G
p_amsterdamer	CPU jobs Amsterdamer group	dml[02-21]	Unlimited	Users in Prof. Amsterdamer's group	Unlimited	Unlimited	192G
p_glickman	GPU jobs Glickman group	dsicsgpu[02–09]	Unlimited	Users in Prof. Glickman's group	Unlimited	Unlimited	192G
p_kraus	GPU jobs Kraus group	dsisarit[02,05]	Unlimited	Users in Prof. Kraus's group	Unlimited	Unlimited	192G
p_weiss	GPU jobs Weiss group	dsiaw15	Unlimited	Users in Prof. Weiss's group	Unlimited	Unlimited	128G

2. Job Submission Examples

Note: All job submission scripts must include lines that begin with #SBATCH. These are special directives that Slurm uses to allocate resources and control job behavior.

This section explains how to create a slurm job script and submit it, including examples and explanations of the parameters.

Basic CPU Job

```
#!/bin/bash
#SBATCH --job-name=cpu_test
#SBATCH --output=cpu_test_%j.out
#SBATCH --error=cpu_test_%j.err
#SBATCH --partition=<Partition-name>
#SBATCH --cpus-per-task=4
#SBATCH --mem=<size>[M- G]
< user commands>
< example: python script.py>
```

Basic GPU Job

```
#!/bin/bash
#SBATCH --job-name=gpu_test
#SBATCH --output=gpu_test_%j.out
#SBATCH --error=gpu_test_%j.err
#SBATCH --partition=<partition name>
#SBATCH --gres=gpu:1
 #SBATCH --mem=<size>[M G]
<user commands>
<example: python gpu_script.py>
```

Resources:

- By default, **1 CPU** is allocated to a job. To request more CPUs use:
  #SBATCH --cpus-per-task=N , where *N* is the number of CPUs you need.
  Ensure that *N* does not exceed the CPU limit of the partition, which can be found in the partition table in Section 1.

- By default, 16 GB of memory is allocated for each CPU assigned to the job.
  To request more memory use: ***#SBATCH --mem=<size>[M- G]***

- By default, **no** GPUs are allocated to a job. To request GPUs use:
  #SBATCH --gres=gpu:1*,* where *N* is the number of GPUs you need.
  Ensure that *N* does not exceed the GPU limit of the partition, which can be found in the partition table in Section 1.

in case the partition includes several types of GPU's and you want a specific GPU, use:
#SBATCH --gres=gpu:<type>:<count>

Email notification:

You can declare (enable) email notifications in your job Slurm script file:
#SBATCH --mail-user=your.email@example.com
#SBATCH --mail-type=[ALL,BEGIN,END,FAIL]

Maximum run time:

You can use the --time option to set the maximum runtime for your job.
#SBATCH --time=HH:MM:SS

- If you don't specify --time, the job will run using the **default time limit of the partition**.
- In case you use --time option, please make sure the time you request **does not exceed** the maximum allowed by your partition.

Docker Job

If you are using docker:

- Run the container in the foreground (**without using -d**).
- Docker container name should be **slurm-$SLURM_JOB_ID** for traceability
  Use this in your batch script:
  DockerName=slurm-job-$SLURM_JOB_ID
  docker run --name "$**DockerName**" --rm my-image:latest python script.py

- If your Slurm job requests **GPUs**, use the list that Slurm provides in **SLURM_JOB_GPUS** variable to restrict Docker to the GPUs allocated to you.
  In your sbatch job script:
  docker run --name "$DockerName" --rm --gpus "device=${**SLURM_JOB_GPUS**}" my-image:latest python script.py
  SLURM_JOB_GPUS is typically a comma-separated list of device indices (e.g., 0,2).

  - If your workflow relies on **CUDA_VISIBLE_DEVICES**, pass it into the container: CUDA_VISIBLE_DEVICES="**${SLURM_JOB_GPUS}**"

Example of a Docker batch script:

```
#!/bin/bash
#SBATCH --job-name=docker_job
#SBATCH --output=docker_job-%j.out
#SBATCH --error=docker_job-%j.err
#SBATCH --partition=generic
#SBATCH --gres=gpu:1
DockerName=slurm-job-$SLURM_JOB_ID
docker run --name "$DockerName" --rm --gpus "device=${SLURM_JOB_GPUS}" \
  my-image:latest python script.py
```

Conda Job

```
#!/bin/bash
#SBATCH --job-name=conda_job
#SBATCH --output=conda_job_%j.out
#SBATCH --error=conda-job_%j.err
#SBATCH --partition=<partition name>
#SBATCH --mem=<size>[M G]
source ~/miniconda3/etc/profile.d/conda.sh
conda activate myenv   # Or source your environment
python script.py
```

MATLAB job

```
#!/bin/bash
#SBATCH --job-name=matlab_job
#SBATCH --output=matlab_job_%j.out
#SBATCH --error=matlab-job_%j.err
#SBATCH --partition=<partition name>
#SBATCH --mem=<size>[M G]
#SBATCH --cpus-per-task=48
<user commands>
matlab -batch "run('mymatlab.m')"
<user commands>
```

Important Note:
When Running MATLAB with **parallel workers** and submitting a MATLAB job with Slurm using:
#SBATCH --cpus-per-task=48 **% <--** 48 is just an example

In order to ensure MATLAB uses **all the CPU cores allocated by Slurm**, add the following to your MATLAB job script (e.g. **mymatlab.m**):

```
num_workers = str2double(getenv('SLURM_CPUS_PER_TASK'));
c = parcluster('local');
c.NumWorkers = num_workers;      % or  exact number of cores you want to use (e.g. 48)
saveProfile(c);
```

3. How to submit the job

1. **Connect to one of the slurm_login servers suing SSH:**
   ssh slurm-login1.lnx.biu.ac.il or
   ssh slurm-login2.lnx.biu.ac.il or
   ssh slurm-login3.lnx.biu.ac.il

2. Submit Your Job**: sbatch myJob.sh**

3. Monitor your job using **squeue -u <your-username>**

4. After submission, Slurm will generate output and error files according to what you have declared in your sbatch job:

   #SBATCH --output=<file-name>.out
   #SBATCH --error=<file-name>.err
   Example:
   #SBATCH --output=matlab_job_%j.out # the file name include the job-id
   #SBATCH --error=matlab-job_%j.err # the file name include the job-id

   Make sure you check output and error files for progress

4. Job Time Limits and Requeue Policy

Jobs submitted with **sbatch** will be automatically **suspended and requeued** by the system when they reach the time limit of the partition they were submitted to (see the Max time in Part 1).

Checkpoints:
To benefit from requeuing, implement **checkpointing** in your application:

- Save progress periodically.
- On restart, resume from the latest checkpoint.

Without checkpointing, the requeued job will start from the beginning.

5. Interactive Jobs

srun is used to run immediate/interactive commands directly on compute nodes managed by Slurm.
srun can both request resources and immediately run a command in that allocation.

Examples:

1. **srun --partition=<partition-name> <command>**
   Runs a single command on a compute node in the specified partition

2. **srun --partition=<partition-name> --pty bash**
   Starts an **interactive shell** on a compute node in the given partition.
   When you exit the shell (exit), the allocation ends and resources are released.

3. **srun --partition=<partition-name> --gres=gpu:1 –time=HH:MM:SS --pty bash**
   Open an interactive shell with 1 GPU for 1 hour.
   When you exit the shell (exit), the allocation ends and resources are released.

Important Notes:

- Interactive jobs launched this way will **not** be automatically requeued.
- Use **srun** only for quick tests, debugging, or development. For longer jobs or those requiring resilience to timeouts, use **sbatch** with checkpointing

6. Private Partitions and Accounts

As noted in Section 1, some partitions are private.
The private partitions **p_kugler**, **p_amsterdamer**, **p_glickman, p_kraus** and **p_weiss** are restricted: only members of the matching group may submit to the matching partition.

- If you belong to Prof. Kugler's group, you may use p_kugler partition.
- If you belong to Prof. Amsterdamer's group, you may use p_amsterdamer partition.
- If you belong to Prof. Glickman's group, you may use p_glickman partition.
- If you belong to Prof. Kraus's group, you may use p_kraus partition.
- If you belong to Prof. Weiss's group, you may use p_weiss partition.

To see which Slurm accounts you belong to use the command:
sacctmgr list assoc where user=$USER format=User,Account -nP
Use your username in place of $USER

Users who are not in the corresponding group cannot use these partitions.

- **Scheduling & limits:** Jobs on these private partitions have **higher scheduling priority** on their dedicated resources and **no MaxTime or resource limits** (per cluster policy).

- **How to submit** a job to private partition
  You should use the matching account, add the following to your batch script:
  #SBATCH --account=<Account-name>
  Account names:
  - For Kuglers group use: ug_kugler as Account-name
  - For Amsterdamer group use: ug_amsterdamer as Account name
  - For Glikman group use: ug_cs_dsi as Account-name
  - For Kraus group use: ug_kraus as Account-name
  - For Weiss group use: ug_weiss as Account-name

  #SBATCH --partition=<p_kugler | p_amsterdamer | p_glickman| p_kraus | p_weiss>

7. Using Jupyter on Slurm Cluster

1. Install Jupyther Notebook locally in your home folder (pip3 install notebook) (if it is not installed already.

2. Start Jupyter (no browser) on the compute node:
   Connect to one of the **slurm_login servers** and run the following command:
   srun --partition=<partition> --gres=gpu:1 jupyter-notebook --no-browser --ip=0.0.0.0 &
   - Note the **node name** and **port number** assigned by Slurm.
   - Copy and save the URL that Jupyter prints (you'll need it later)

   output example:
   http://nodename:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
   http://127.0.0.1:8888/tree?token=e34d73f70fa48254500e0556663db7b859985448f9d9829d
   In this example **node name** and **port number,** assigned by Slurm, are: nodename and 8888

3. On your local PC - Create an SSH tunnel:
   On your local PC, run:
   ssh -L <port-num>:localhost:<port-num> -J <username>@<slurm_login_server> <username>@<node>
   - Replace <port-num> with the actual port number from step 1.
   - Replace <username> with your username.
   - Replace <node> with the node assigned by Slurm (from step 1).

Use Jupyter Notebook from a Web Browser:

paste the copied Jupyter URL (from step 1, second line) into your **local web browser**.
You should now see the notebook interface running on the remote Slurm node.

Use Jupyter Notebook inside VS Code:

1. On your local PC, open VS Code and install the Microsoft **Python** and **Jupyter** extensions (if they are not already installed). (No Remote-SSH is needed)
2. Press Ctrl+Shift+P → type "Jupyter: Specify Jupyter Server URL"
3. Choose "Existing". Paste the URL printed by Jupyter (the one that you copied when started jupyter) for example: http://localhost:8888/?token=abcd1234...
   VS Code will now connect directly to that remote Jupyter kernel running on the Slurm node.

To create a new notebook file:

- Ctrl + Shift + P → Jupyter: Create New Jupyter Notebook
- On the top-right corner and click Select Kernel.
- Choose → Existing Jupyter Server…
  Paste the remote Jupyter server URL the one that you copied when started jupyter) for example: http://localhost:8888/?token=abcd1234...)
  VS Code will now connect to the Jupyter Notebook running on your Slurm node.

Important Notes:

- After testing and debugging your code, submit the **actual job** to the desired partition using **sbatch** and a Slurm **batch script**.
- It is recommended to perform **debugging and testing** in a partition with **less powerful CPU/GPU nodes**, to avoid occupying high-demand resources during development. For **production jobs**, use sbatch with sbatch scripts.
- Using –time=HH:MM:SS will make sure the Job will stop automatically when the --time limit is reached or the partition's time limit is hit.

8. Useful Slurm Commands

Submit a job	***sbatch job.sh***
Check your jobs	***squeue -u $USER***
Cancel a job	***scancel <job_id>***
View available partitions	***sinfo***

Version 1.6, 3 Nov 2025