Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

BEN-Cluster: Users Operational Guide

Francesco Gargiulo, Emilio Greco

RT-ICAR-NA-2024-03 **September 2024**

BEN-Cluster: Users Operational Guide

Francesco Gargiulo, Emilio Greco
Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche (ICAR-CNR)
Via Pietro Castellino, 111 – 80131 Napoli, Italia
{francesco.gargiulo, emilio.greco}@icar.cnr.it

Abstract

Introducing advanced hardware infrastructure is pivotal for conducting cutting-edge scientific international research. To address this need, the Institute of High-Performance Computing and Networks (ICAR) of the National Research Council (CNR) and other CNR Research Institutes have acquired the BEN-Cluster, an IBM-produced high-performance computing cluster. This cluster is designed to support researchers in a wide range of experiments that require substantial computational power, particularly those in fields such as High-Performance Computing (HPC), Artificial Intelligence (AI), Big Data Analytics, and cybersecurity.

The BEN-Cluster is tailored to handle complex AI models, especially those developed through deep learning paradigms, by leveraging GPUs' acceleration capabilities. Its infrastructure is also ideal for processing and analyzing large datasets, which are increasingly common in scientific research. Furthermore, the cluster's robust architecture supports the simulation and validation of methodologies that demand extensive computational resources.

This operational guide aims to provide a comprehensive overview of the BEN-Cluster, outlining its hardware specifications, job management through the Platform Load Sharing Facility (LSF), and the setup and management of software environments using Conda. Detailed instructions on simulating a workstation environment via VNC are also included, facilitating a seamless transition for researchers from their local machines to the cluster environment.

Through this document, authorized users will gain the necessary insights and technical knowledge to maximize the use of the BEN-Cluster. By understanding the operational protocols and best practices, researchers can efficiently harness the cluster's capabilities to advance their scientific inquiries. Future developments and potential enhancements to the infrastructure are also explored, aiming to further improve the cluster's performance and alignment with regulatory requirements.

1. Hardware Specifications

The BEN-Cluster, installed at ICAR-CNR’s CED, is a state-of-the-art high-performance computing system designed to support various scientific research applications. This section provides a detailed overview of the hardware components and their specifications and benchmarking data that highlights this infrastructure's performance advantages.

Cluster Overview

The BEN-Cluster comprises 73 compute nodes with IBM POWER9 processors and NVIDIA Tesla V100 GPUs. The total system configuration includes 62 TB of RAM and 1.6 PB of disk space, segmented into various types of storage to optimize performance across different workloads.

The cluster's architecture is designed for high computational throughput, scalability, and flexibility. It is ideal for demanding scientific applications such as HPC, AI, ML, Big Data Analytics, and cybersecurity. Integrating IBM's POWER9 processors and NVIDIA's Tesla V100 GPUs provides a robust platform that can easily handle intensive parallel processing tasks, large-scale simulations, and real-time data processing.

Key Features and Capabilities

- **High Computational Power:** The cluster's 73 nodes, equipped with POWER9 processors and Tesla V100 GPUs, offer high computational throughput, essential for complex simulations and data-intensive tasks.
- **Scalability:** The architecture supports scalable expansion, allowing more nodes and storage to be added as research demands grow.
- **Advanced Networking:** A 100 Gb/s Infiniband network ensures high-speed, low-latency communication between nodes, which is crucial for distributed computing applications.
- **Optimized Memory Hierarchy:** With 62 TB of RAM, the cluster supports large in-memory computations and efficient data access, reducing bottlenecks in data-intensive workflows.
- **Flexible Storage Solutions:** The hierarchical storage system, comprising SSDs, NVMe, and HDDs, provides a balanced approach to storage, combining speed and capacity to handle various data access patterns.

Detailed Node Specifications

25 IBM SYSTEM POWER AC922 Compute Nodes:

- **Processors:** 2 x 16-core 2.7 GHz (3.3 GHz Turbo) POWER9
- **Storage:** 2 x 1.92TB SATA/SSD disks
- **GPUs:** 4 x NVIDIA Tesla V100 16GB
- **Memory:** 512 GB RAM (16 x 32 GB DDR4)

44 IBM SYSTEM POWER AC922 Compute Nodes:

- **Processors:** 2 x 16-core 2.7 GHz (3.3 GHz Turbo) POWER9
- **Storage:** 2 x 1.92TB SATA/SSD disks
- **GPUs:** 4 x NVIDIA Tesla V100 32GB
- **Memory:** 1024 GB RAM (16 x 64 GB DDR4)

4 IBM SYSTEM POWER AC922 Compute Nodes:

- **Processors:** 2 x 16-core 2.7 GHz (3.3 GHz Turbo) POWER9
- **Storage:** 2 x 1.92TB SATA/SSD disks, 2 x 6.4 TB NVMe Flash disks
- **GPUs:** 4 x NVIDIA Tesla V100 32GB
- **Memory:** 1024 GB RAM (16 x 64 GB DDR4)

Storage and Network Configuration

The BEN-Cluster features a hierarchical storage architecture:

- **284 TB SSD** for high-speed data access and intermediate storage.
- **51 TB NVMe** for ultra-fast, low-latency storage requirements.
- **1.3 PB HDD** for long-term data storage and archival purposes.

A 100 Gb/s Infiniband connection facilitates the network within the cluster, ensuring low-latency, high-throughput communication between nodes, which is critical for parallel processing tasks and large-scale simulations.

Benchmarking and Performance Evaluation

Several benchmarks were conducted to evaluate the BEN-Cluster's performance, comparing its capabilities to similar systems based on the Intel architecture. The results, illustrated in the following, demonstrate significant performance improvements in memory management and GPU utilization.

1. **Deep Learning Workloads:** The POWER9 processors, combined with the Tesla V100 GPUs, show a 50% increase in training speeds for deep learning models compared to Intel-based systems.
2. **Big Data Analytics:** The BEN-Cluster's optimized memory hierarchy and storage subsystems make data processing tasks 30% faster.
3. **Security Applications:** Blockchain and encryption-related computations exhibit a 40% efficiency gain, highlighting the cluster's superior handling of complex, data-intensive operations.

For further details on the hardware architecture and performance benchmarks, refer to the following sources:

1. **IBM Power Systems Technical White Paper** - A comprehensive guide on the POWER9 architecture and its applications in high-performance computing. Available at: IBM Technical Papers.

2. **NVIDIA Tesla V100 GPU Overview**—The Tesla V100 GPU's detailed specifications and performance metrics are available at NVIDIA Tesla V100[1].
3. **High-Performance Computing with IBM POWER9 and NVIDIA GPUs** - A study on integrating IBM POWER9 processors with NVIDIA GPUs in HPC environments. Available at: HPC Wire[2].

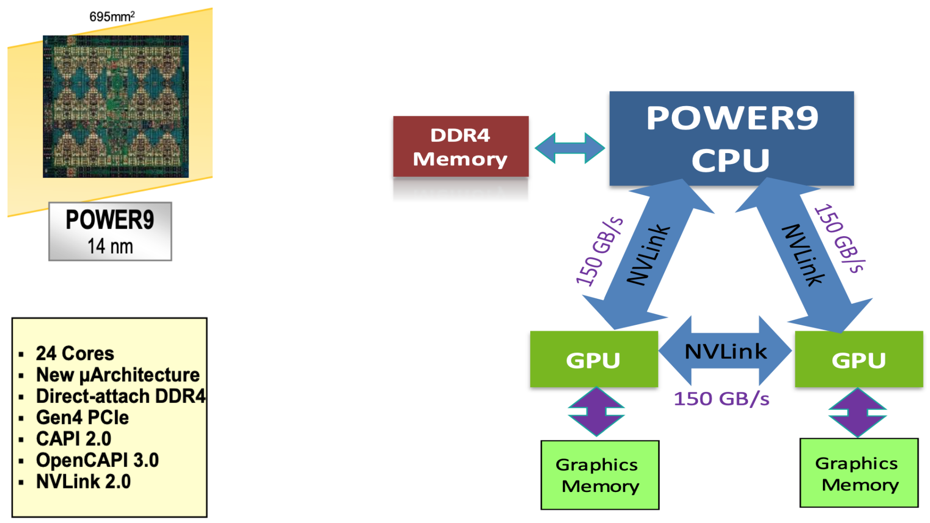
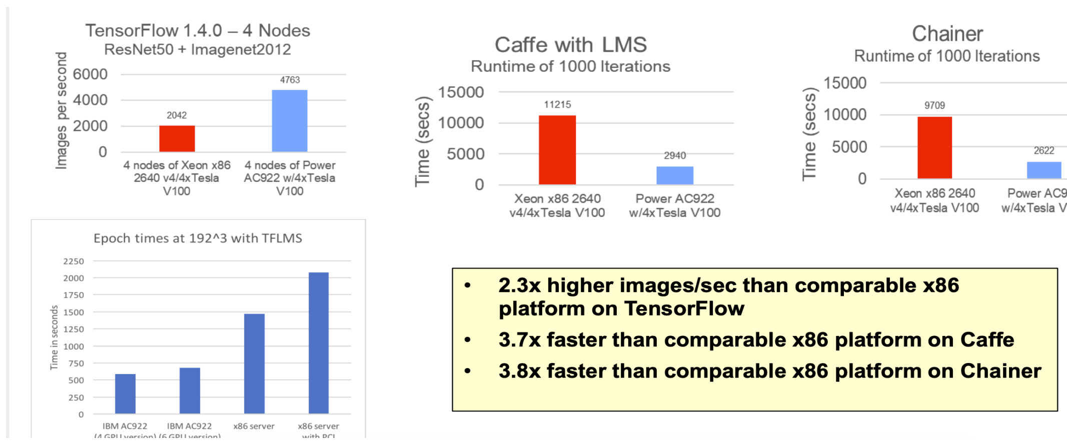


Figure 1: a) Performance comparison of POWER9 vs. Intel architectures on deep learning tasks. b) Data throughput and latency benchmarks for storage configurations.



- **2.3x higher images/sec than comparable x86 platform on TensorFlow**
- **3.7x faster than comparable x86 platform on Caffe**
- **3.8x faster than comparable x86 platform on Chainer**

Figure 2: Efficiency metrics for blockchain computations on BEN-Cluster.

2. Job management through LSF

The BEN-Cluster utilizes the Platform Load Sharing Facility (LSF) to manage and schedule jobs. LSF is a powerful job scheduler and workload management platform that allows users to efficiently allocate and utilize computational resources within a cluster environment. This section provides an

[1] https://www.nvidia.com/en-us/data-center/tesla-v100/
[2] https://www.hpcwire.com/

overview of LSF's functionalities, how to manage jobs, and best practices for using the BEN-Cl LSF Job Management.

The Platform Load Sharing Facility (LSF) is the job manager used by the BEN-Cluster. All documentation and commands are available through the IBM website. This section lists the key concepts and functionalities that will allow a quick first use of the tool.

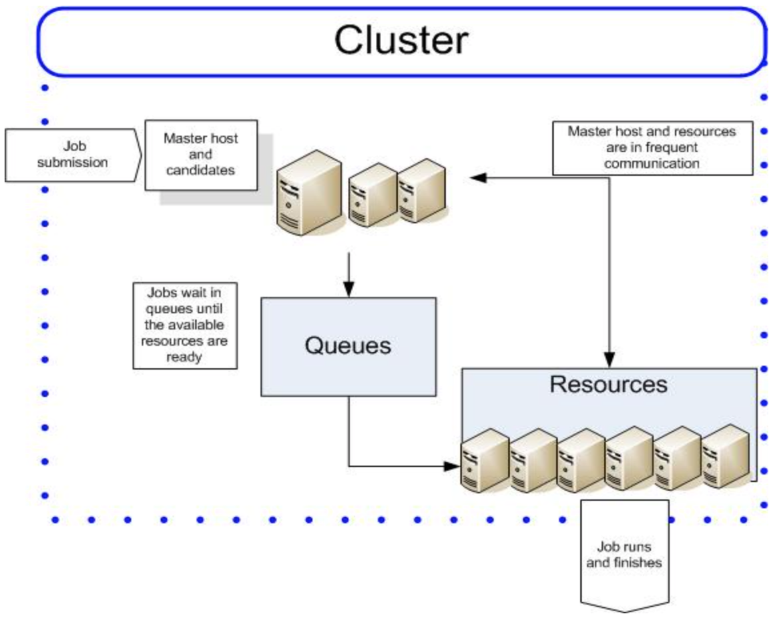
Job submission occurs through the JOB_MASTER, a frontend process running on the master hosts named **ems1** and **ben-login**. The JOB_MASTER places the request in one of the queues, waiting for the required resources to become available for the proper job execution. Once resources are available, the job is launched on a subset of the requested cluster resources.

Resources are allocated based on the requests specified by the user at the time of job submission. These requests may include the number of cores, amount of memory, estimated execution time, and other specifications. LSF uses advanced scheduling algorithms to optimize resource allocation based on these requests and cluster policies.

When a job ends, it returns the output to the JOB_MASTER. It's important to note that jobs don't always terminate successfully; they can end for various reasons, including execution errors, exceeded time limits, or manual interruptions. LSF provides mechanisms to monitor job status and notify users of their completion or any issues. Users can use specific LSF commands such as `bjobs` to check the status of their jobs or configure email notifications that are automatically sent upon job completion.

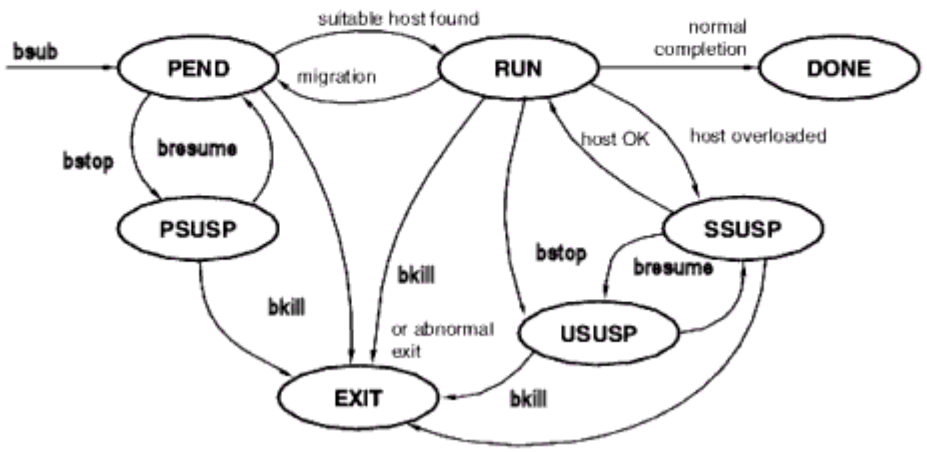
Throughout all phases, the JOB_MASTER constantly communicates with the resources, monitoring each one's status and making this information available to cluster users. This allows for efficient resource management and gives users a real-time view of cluster usage.

Fig.2 illustrates the flow diagram of job processing, visually showing the phases from submission to execution and completion of the job within the LSF system.



Cluster
Job submission
Master host and candidates
Master host and resources are in frequent communication
Jobs wait in queues until the available resources are ready
Queues
Resources
Job runs and finishes

The diagram below describes the states where jobs can be found during the request management phases, highlighting the nature of transitions between the different possible states.



Below is a brief description of the main states of a job in LSF:

- **PEND:** This is the state of a job submitted through the `bsub` command. The process is waiting to be executed on an appropriate queue. In this state, LSF evaluates available resources and priorities to determine when and where to run the job.
- **RUN:** Indicates that the job runs on one or more cluster nodes. The requested resources have been allocated, and the job is processing data.
- **DONE:** This state signals that the job has been completed successfully. The output is available for the user to retrieve.

- **PSUSP (Pre-execution Suspended):** The system suspended the job before the execution began. This can happen for various reasons, such as scheduling policies or administrative interventions.
- **EXIT:** This indicates that the job has terminated abnormally, usually due to an error during execution or exceeding assigned resource limits.
- **USUSP (User Suspended):** The job has been suspended by the user who submitted it using the `bstop` command.
- **SSUSP (System Suspended):** The system has suspended the job, generally for resource management or cluster maintenance.

Users can intervene with various commands to force the state transition of their jobs in LSF. Below are the main commands and how they affect job states:

- **bresume [job_ID]:** Resumes the execution of a suspended job, for example, from USUSP (User Suspended) or PSUSP (Pre-execution Suspended) to RUN or PEND (depending on resource availability).
- **bstop [job_ID]:** Suspends a running job, for example, from RUN state to USUSP (User Suspended) state.
- **bkill [job_ID]:** Terminates a job in any state (RUN, PEND, USUSP, SSUSP, PSUSP), putting it in the EXIT state.
- **brequeue [job_ID]:** Puts a job back in the queue to be reprocessed from RUN, USUSP, SSUSP to PEND.
- **bswitch [queue_name] [job_ID]:** Moves a job from one queue to another, for example, from PEND to PSUSP.
- **bmod [options] [job_ID]:** Modifies the parameters of a pending job.

The command `bqueues` display the status of the various queues on the BEN-Cluster. Below is an example output:

```
QUEUE_NAME   PRIO STATUS     MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
interactive  30   Open:Active -    -    -    -     12    4    8    0
ext_batch    20   Open:Active -    -    -    -     50   30   20    0
icar_batch   20   Open:Active -    -    -    -     40   20   20    0
admin        50   Open:Active -    -    -    -      2    0    2    0
dataq        40   Open:Active -    -    -    -      3    1    2    0
```

This output provides a comprehensive overview of the current status of the queues, including:

- **QUEUE_NAME:** The name of the queue.
- **PRIO:** The priority of the queue.
- **STATUS:** The current status of the queue.
- **MAX:** The maximum number of jobs allowed in the queue.
- **JL/U:** The job limit per user.

- **JL/P:** The job limit per processor.
- **JL/H:** The job limit per host.
- **NJOBS:** The number of jobs currently in the queue.
- **PEND:** The number of pending jobs.
- **RUN:** The number of running jobs.
- **SUSP:** The number of suspended jobs.

This output shows the status of various queues, including the queue name, priority, status, job limits, number of jobs, pending jobs, running jobs, and suspended jobs. Each queue is listed with its current status and job details, providing a comprehensive overview of the job distribution and activity within the cluster.

BEN-Cluster: LSF Queues

The BEN-Cluster is configured with five distinct queues to cater to different types of job submissions and user requirements:

- **interactive (default queue):** This queue is intended for interactive jobs that require user input and real-time monitoring. It allows multiple users to share the same node resources.

  - This is set as the default queue.
  - It's the only queue where interactive jobs can be run.
  - Interactive jobs are launched with the `-Is` option.
  - If no specific queue is indicated during job submission, the job will automatically be processed in this queue.
  - Jobs running in this queue cannot reserve a node exclusively, allowing for better resource sharing.
  - This queue is ideal for interactive tasks requiring user input or real-time monitoring.

- **ext_batch:** This queue is designed for batch jobs submitted by non-ICAR users. It supports exclusive node usage, making it suitable for long-running, resource-intensive tasks.

  - This queue is designed to run batch jobs submitted by non-ICAR users.
  - It allows for exclusive node usage, meaning a job can request and use an entire node without sharing resources with other jobs.
  - Suitable for longer-running, resource-intensive tasks that don't require user interaction.

- **icar_batch:** This is similar to ext_batch but specifically allocated for ICAR users. It also supports exclusive node usage and may have different resource limits or priorities tailored to ICAR requirements.

  - Similar to ext_batch, but specifically allocated for ICAR (Institute for Calculus Applications and Research) users.
  - It also allows for exclusive node usage.
  - It may have different resource limits or priorities than ext_batch, which is tailored to ICAR requirements.

- **admin:** Reserved for administrative tasks, this queue likely has higher priorities and fewer restrictions to allow system administrators to perform necessary maintenance and management tasks.

    - Limited to administrative tasks.
    - It likely has higher priorities and fewer restrictions to allow system administrators to perform necessary maintenance and management tasks.

- **dataq:** Also restricted to administrative tasks, this queue is dedicated to data management operations such as backups, data transfers, or other storage-related tasks.

    - Also limited to administrative tasks.
    - Possibly dedicated to data management operations, such as backups, data transfers, or other storage-related tasks.

Using the correct queue is essential to allow efficient resource allocation based on the nature of the jobs and the users submitting them. It ensures that different types of computational tasks can be accommodated while maintaining fair access and optimal utilization of the cluster resources. In details:

- Batch queues are designed for jobs that can run unattended, often for longer periods. They typically have particular resource requests and can have the necessity of exclusive node access, making them suitable for computationally intensive tasks;
- The interactive queue, on the other hand, is optimized for shorter, user-interactive sessions. It usually doesn't need specific resource requests, and exclusive node access is not allowed to simultaneously ensure resource availability for multiple users.

BEN-Cluster: Practical Use Cases

Interactive jobs are ideal for real-time tasks that require user interaction. The commands for launching an interactive job are as follows:

Without GPU support:

```
bsub -q interactive -Is /bin/bash
```

With GPU support:

```
bsub -q interactive -gpu "num=1:gtile=1" -Is /bin/bash
```

Since the "interactive" queue is the default, the -q interactive parameter is optional and can be omitted. These commands open a shell session on the cluster, allowing users to execute commands interactively.

Batch job submissions are suitable for non-interactive tasks that can run unattended. Depending on the user's affiliation with ICAR, there are two specific queues for batch job submission: icar_batch for ICAR users and ext_batch for external users. Both queues function identically in terms of job handling.

Command to submit a batch job:

```
bsub -q icar_batch -gpu "num=1:gtile=1" -o logs/log.out -e logs/log.err “bash test_bsub.sh"
```

- **-q icar_batch:** Specifies the queue for the job. Use ext_batch if you are an external user.
- **-gpu "num=1:gtile=1"**: Requests one GPU for the job
- **-o logs/log.out:** Redirects the job's standard output to logs/log.out.
- **-e logs/log.err:** Redirects the job's standard error to logs/log.err.

These redirection parameters are crucial as they help capture the job's output and error logs for later analysis.

For complete information about the GPU options, refer to the documentation on GPU management. Below is a synopsis of the options available for the -gpu parameter:

```
bsub -gpu - | "[num=num_gpus[/task | host]] [:mode=shared | exclusive_process]

[:mps=yes[,shared][,nocvd] | no | per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd]]

[:j_exclusive=yes | no] [:aff=yes | no] [:block=yes | no] [:gpack=yes | no]

[:gmodel=model_name[-mem_size]] [:gtile=tile_num|'!'] [:gmem=mem_value] [:nvlink=yes]"
```

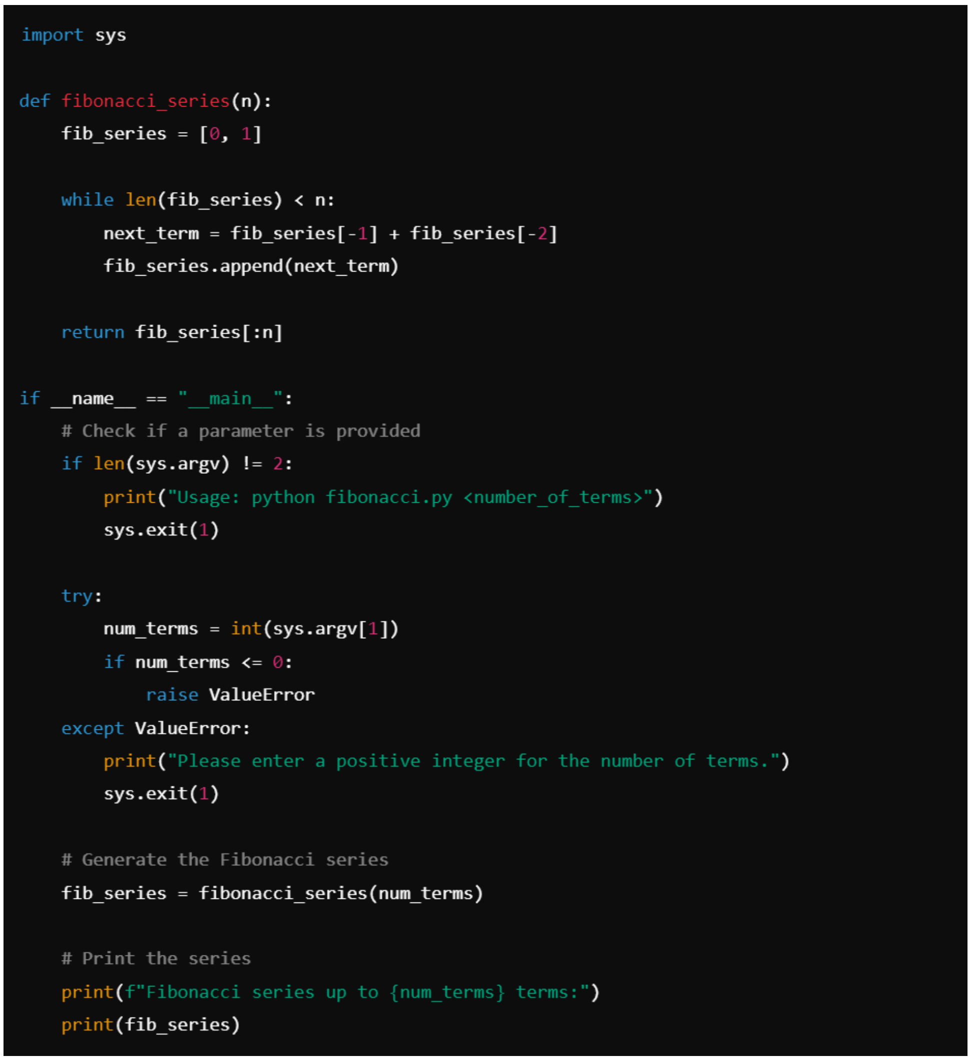
Explanation of -gpu Options:

- **num:** Specifies the number of GPUs required.
  - Example: num=1 for one GPU.
- **mode:** Determines the mode of GPU usage.
  - shared: The GPU can be shared among multiple processes.
  - exclusive_process: The GPU is exclusively used by one process.
- **gmem:** Specifies the amount of GPU memory.
  - Example: gmem=4G for 4GB of GPU memory.
- **mps:** Enables NVIDIA Multi-Process Service (MPS) with optional configurations.

- **j_exclusive:** Determines job exclusivity on GPUs.
- **aff:** GPU affinity settings.
- **block:** Blocks jobs until the requested GPU resources are available.
- **gpack:** GPU packing strategy.
- **gmodel:** Specifies the GPU model and, optionally, the memory size.
- **gtile:** Specifies GPU tiling settings.
- **nvlink:** Enables or disables NVLink for GPU communication.

For example, suppose you have a Python script named `fibonacci.py` that you want to run within a specific Conda environment, testing various values for the script's parameters. Here is a step-by-step guide to submitting this job:

1. **Prepare the Batch Script:** Create a script (e.g., run_fibonacci.sh) with the following content:



```python
import sys

def fibonacci_series(n):
    fib_series = [0, 1]

    while len(fib_series) < n:
        next_term = fib_series[-1] + fib_series[-2]
        fib_series.append(next_term)

    return fib_series[:n]

if __name__ == "__main__":
    # Check if a parameter is provided
    if len(sys.argv) != 2:
        print("Usage: python fibonacci.py <number_of_terms>")
        sys.exit(1)

    try:
        num_terms = int(sys.argv[1])
        if num_terms <= 0:
            raise ValueError
    except ValueError:
        print("Please enter a positive integer for the number of terms.")
        sys.exit(1)

    # Generate the Fibonacci series
    fib_series = fibonacci_series(num_terms)

    # Print the series
    print(f"Fibonacci series up to {num_terms} terms:")
    print(fib_series)
```

2. **Submit the Job:** Submit the batch job using the following command:

Starting from this script, we have to write another one (`run_bsub.sh`) to instantiate the environment on the destination node, such as:

```
#!/bin/bash
echo "job start with parameters $1"
. ~/.bashrc
conda activate base
cd /home/users/gargiulo/PycharmProjects/example
python fibonacci.py $1
echo "job finished with parameters $1"
```

Note that we have considered the input parameter `$1` for the `fibonacci.py` script.

```
bsub -q icar_batch -gpu "num=1:gtile=1" -o logs/log.out -e logs/log.err "bash run_fibonacci.sh 10 "
```

This command will run `fibonacci.py` within the specified Conda environment with parameters set to 10. The standard output and error logs will be saved in `logs/log.out` and `logs/log.err` for further review.

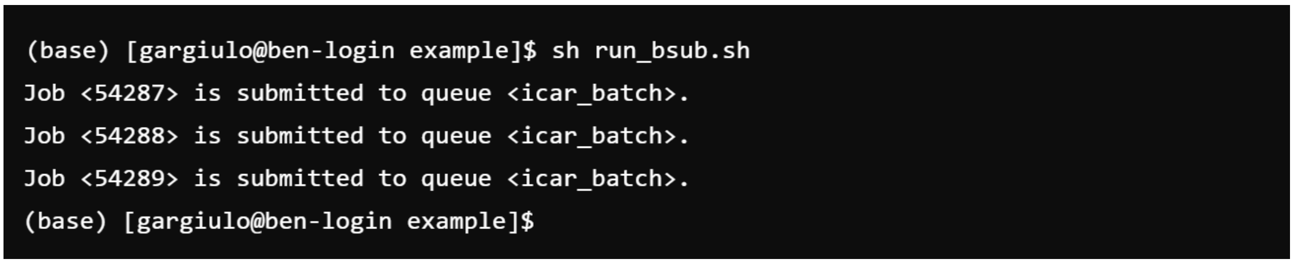
We must prepare and launch this final script to test the Fibonacci script for n={10,20,30} in parallel on the cluster nodes.

```
#!/bin/bash
gpu_option="num=1:gtile=1"
n_rep=("10" "20" "30")
for n in ${n_rep[@]}; do
    bsub -q icar_batch -gpu $gpu_option -o log/fibonacci_${n}.out \
    -e log/fibonacci_${n}.err "bash run.sh ${n}"
done
```

The final project's folder has to contain at least these files/directories:

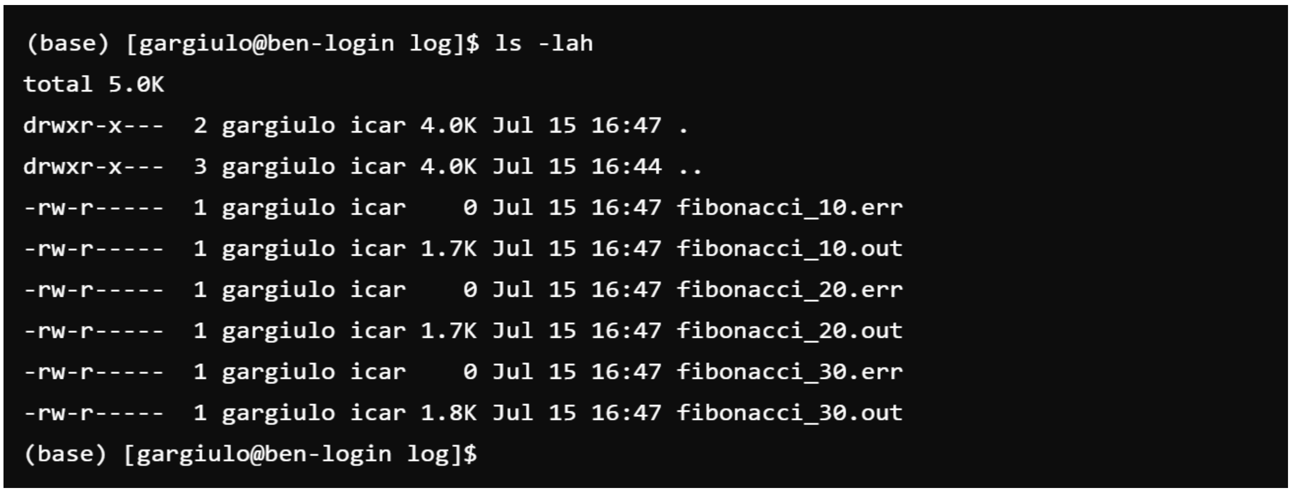
```
(base) [gargiulo@ben-login example]$ ls -lah
total 6.0K
drwxr-x---  3 gargiulo icar 4.0K Jul 15 16:44 .
drwxr-x--- 48 gargiulo icar 4.0K Jul 15 16:07 ..
-rw-r-----  1 gargiulo icar  797 Jul 15 16:33 fibonacci.py
drwxr-x---  2 gargiulo icar 4.0K Jul 15 16:45 log
-rw-r-----  1 gargiulo icar  206 Jul 15 16:44 run_bsub.sh
-rw-r-----  1 gargiulo icar  191 Jul 15 16:26 run.sh
(base) [gargiulo@ben-login example]$
```

Note that we have created the sub-folder "log" to put all the batch output.
Now, it is possible to launch the batch as follows:



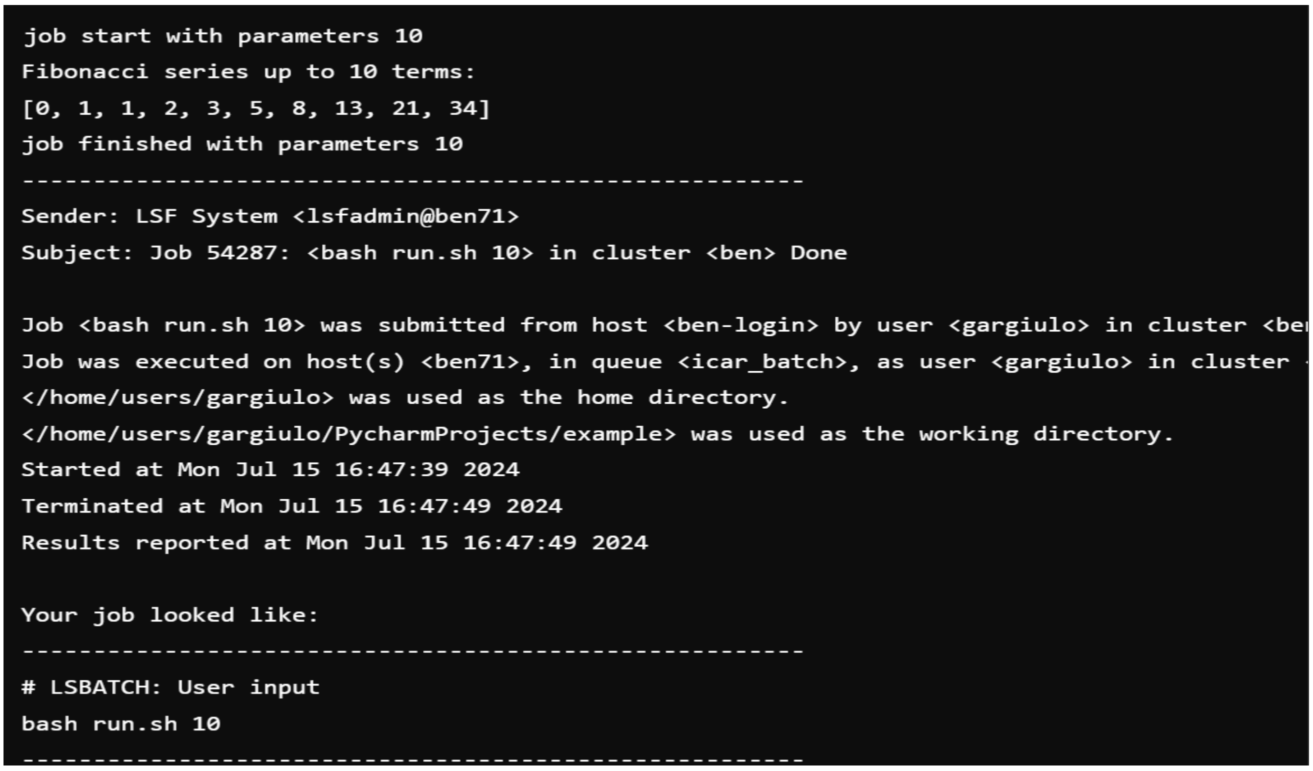
```
(base) [gargiulo@ben-login example]$ sh run_bsub.sh
Job <54287> is submitted to queue <icar_batch>.
Job <54288> is submitted to queue <icar_batch>.
Job <54289> is submitted to queue <icar_batch>.
(base) [gargiulo@ben-login example]$
```

The subfolder "log" contains the outputs of the jobs, as you can see in the following screenshot:



```
(base) [gargiulo@ben-login log]$ ls -lah
total 5.0K
drwxr-x---  2 gargiulo icar 4.0K Jul 15 16:47 .
drwxr-x---  3 gargiulo icar 4.0K Jul 15 16:44 ..
-rw-r-----  1 gargiulo icar    0 Jul 15 16:47 fibonacci_10.err
-rw-r-----  1 gargiulo icar 1.7K Jul 15 16:47 fibonacci_10.out
-rw-r-----  1 gargiulo icar    0 Jul 15 16:47 fibonacci_20.err
-rw-r-----  1 gargiulo icar 1.7K Jul 15 16:47 fibonacci_20.out
-rw-r-----  1 gargiulo icar    0 Jul 15 16:47 fibonacci_30.err
-rw-r-----  1 gargiulo icar 1.8K Jul 15 16:47 fibonacci_30.out
(base) [gargiulo@ben-login log]$
```

If we open one of them, such as "`fibonacci_10.out`", the output is the following:



```
job start with parameters 10
Fibonacci series up to 10 terms:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
job finished with parameters 10
------------------------------------------------------
Sender: LSF System <lsfadmin@ben71>
Subject: Job 54287: <bash run.sh 10> in cluster <ben> Done

Job <bash run.sh 10> was submitted from host <ben-login> by user <gargiulo> in cluster <be
Job was executed on host(s) <ben71>, in queue <icar_batch>, as user <gargiulo> in cluster
</home/users/gargiulo> was used as the home directory.
</home/users/gargiulo/PycharmProjects/example> was used as the working directory.
Started at Mon Jul 15 16:47:39 2024
Terminated at Mon Jul 15 16:47:49 2024
Results reported at Mon Jul 15 16:47:49 2024

Your job looked like:
------------------------------------------------------
# LSBATCH: User input
bash run.sh 10
------------------------------------------------------
```

```
-------------------------------------------------------
Successfully completed.
Resource usage summary:
CPU time: 0.81 sec.
Max Memory: 19 MB
Average Memory: 15.50 MB
Total Requested Memory: -
Delta Memory: -
Max Swap: -
Max Processes: 5
Max Threads: 8
Run time: 6 sec.
Turnaround time: 11 sec.
The output (if any) is above this job summary.
PS:
Read file <log/fibonacci_10.err> for stderr output of this job.
```

The first line contains the proper output; after that, the job output provides a wealth of information, including:

- **CPU Time**: The amount of processor time consumed by the job.
- **Max Memory**: The peak memory usage during the job execution.
- **Job Submission Details**: Hostname, username, queue, and other metadata.

This detailed information can be useful for optimizing and debugging jobs. For further customization and advanced options, refer to the official documentation.

Monitoring Jobs and Cluster Resources

To view the execution status of a particular job, it is sufficient to run the command:

```
bjobs [job_ID]
```

If a specific ID is not specified, the command will show the status of all jobs for the current user. The command output will display a series of information similar to the following:

```
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
12345   user1   RUN   icar_batch node01      node02      job_name   Jul 20 10:00
```

This output provides crucial information such as the job ID, the user who submitted it, the current status, the queue it's in, the host from which it was submitted, the host on which it's running (if applicable), the job name, and the submission time. To obtain more detailed information about a specific job, you can use the command:

```
bjobs -l [job_ID]
```

This will provide a complete job report, including the resources used, execution times, and other relevant information.

```
Job <123456>, User <user1>, Project <default>, Status <RUN>, Queue <normal>,
Command </my_simulation>
Tue Jun 15 10:30:15: Submitted from host <submit_host>, CWD </home/user1/project>;
Tue Jun 15 10:32:20: Started on <node001>, Execution Home </home/user1>, Execution
CWD </home/user1/project>;

SCHEDULING PARAMETERS:
r15s   r1m    r15m   ut     pg     io     ls     it     tmp    swp    mem
-      -      -      -      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -      -      -      -

RESOURCE REQUIREMENT DETAILS:
Combined: select[type == any] order[r15s:pg] rusage[mem=4096,swp=2048]
Effective: select[type == any] order[r15s:pg] rusage[mem=4096,swp=2048]

RESOURCE USAGE:
CPU time : 2:13:43.00
MEM usage : 3789.30 MB
SWAP usage: 1024.00 MB

SCHEDULING PARAMETERS:
r15s   r1m    r15m   ut     pg     io     ls     it     tmp    swp    mem
-      -      -      -      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -      -      -      -

EXTERNAL MESSAGES:
MSG_ID FROM  POST_TIME    MESSAGE                    ATTACHMENT
0      user1 Jun 15 11:30 Job is running smoothly    N

PARALLEL JOB INFORMATION:
Current Hosts: node001 (1 slots)
```

LSF provides several commands to monitor the status of jobs and the utilisation of cluster resources:

- **bjobs:** Displays the status of jobs submitted by the user.

```
bjobs
```

Example: Output of bqueues

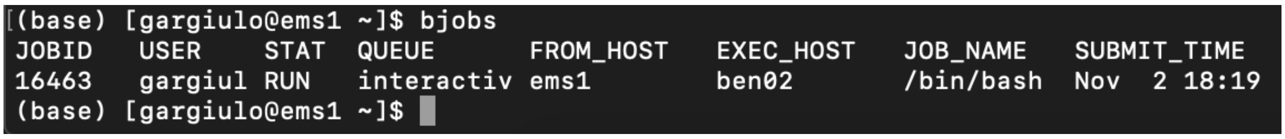
The command `bqueues` display the status of the various queues on the BEN-Cluster. Below is an example output:

```
QUEUE_NAME   PRIO STATUS     MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
interactive  30   Open:Active  -    -    -    -    12    4    8    0
ext_batch    20   Open:Active  -    -    -    -    50   30   20    0
icar_batch   20   Open:Active  -    -    -    -    40   20   20    0
admin        50   Open:Active  -    -    -    -     2    0    2    0
dataq        40   Open:Active  -    -    -    -     3    1    2    0
```

This output shows the status of various queues, including the queue name, priority, status, job limits, number of jobs, pending jobs, running jobs, and suspended jobs. Each queue is listed with its current status and job details, providing a comprehensive overview of the job distribution and activity within the cluster.

Example: Output of bjobs

The command `bjobs` shows the status of jobs submitted by the user. Below is an example output:



```
(base) [gargiulo@ems1 ~]$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
16463   gargiul RUN   interactiv ems1        ben02       /bin/bash  Nov  2 18:19
(base) [gargiulo@ems1 ~]$
```

The output of the `bjobs` command shows that the process is in the RUNNING state on the host ben02 and is executing the /bin/bash process. It is important to note that this command only displays active jobs for the current user and not all jobs running on the system.

This output provides crucial information such as:

- **JOBID:** The ID of the job.
- **USER:** The user who submitted the job.
- **STAT:** The current status of the job.
- **QUEUE:** The queue in which the job is placed.
- **FROM_HOST:** The host from which the job was submitted.
- **EXEC_HOST:** The host on which the job is running.
- **JOB_NAME:** The name of the job.
- **SUBMIT_TIME:** The time at which the job was submitted.

- **lsload:** Provides an overview of resource utilisation across the entire cluster.

```
lsload
```

- **lsload -gpuload:** Specifically monitors GPU utilisation.

```
lsload -gpuload
```

Job Management Commands:

- **bresume [job_ID]:** Resumes executing a suspended job.
- **bstop [job_ID]:** Suspends a running job.
- **bkill [job_ID]:** Terminates a job in any state.
- **brequeue [job_ID]:** Requeues a job to be reprocessed.
- **bswitch [queue_name] [job_ID]:** Moves a job from one queue to another.
- **bmod [options] [job_ID]:** Modifies the parameters of a pending job.

These commands allow users to manage their jobs efficiently, ensuring optimal use of cluster resources and minimizing downtime due to job errors or resource contention.

Killing Jobs

Suppose you have started a process that is not functioning correctly. To stop it, LSF provides the `bkill` command. Here is how you can use it:

1. Identify the Job ID (JOBID): First, you need to find the JOBID of the problematic job. You can do this using the `bjobs` command to list all your running jobs.
2. Kill the Job: Once you have identified the JOBID, you can kill the job using the following command:

```
bkill JOBID
```

For example, to terminate a job with a JOBID of 12345, you would execute the following:

```
bkill 12345
```

In a more complex scenario where you have generated hundreds of processes that are not functioning correctly, you can use the `bkill` command to kill all jobs simultaneously. This is done using:

```
bkill 0
```

More in detail:

- **`bjobs`:** This command lists all your active jobs, their JOBIDs, and statuses. It helps you identify which jobs need to be terminated.
- **`bkill` JOBID:** This command terminates a specific job identified by its JOBID. It sends a signal to stop the job immediately.
- **`bkill` 0:** This command kills all the jobs the user submits. It is useful when you need to clear out many problematic jobs quickly.

Imagine you have submitted multiple jobs to test a script and realise an error is causing all the jobs to hang or produce incorrect results. Instead of killing each job individually, you can use `bkill 0` to terminate all of them in one go, allowing you to correct the error and resubmit the jobs more efficiently.

Using these commands effectively ensures you can manage and control your job submissions, minimize resource wastage, and improve overall system performance. For more detailed options and advanced usage, refer to the official LSF documentation.

3. Conda Repository Management

The ANACONDA3[3] package manager is installed and configured on the BEN Cluster. Using this package manager, conda environments can be created, and all the packages necessary to develop or use a program can be installed. Specifically, the first thing to configure is the channels. For the PowerPC architecture, IBM has defined its channels with distributions of packages specifically modified to optimize the cluster architecture.

Configuring Conda Channels

One of these channels is:

```
https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda/
```

To configure this channel in the system, you can use the following command (note that it is already configured by default on the platform, so this step is not necessary):

```
conda config --prepend channels https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda/
```

[3] https://docs.anaconda.com/

Creating and Activating Conda Environments

To create a conda environment with everything needed to operate, use the following syntax for creating and activating the environment:

```
conda create --name <environment_name> python=<python_version>
conda activate <environment_name>
```

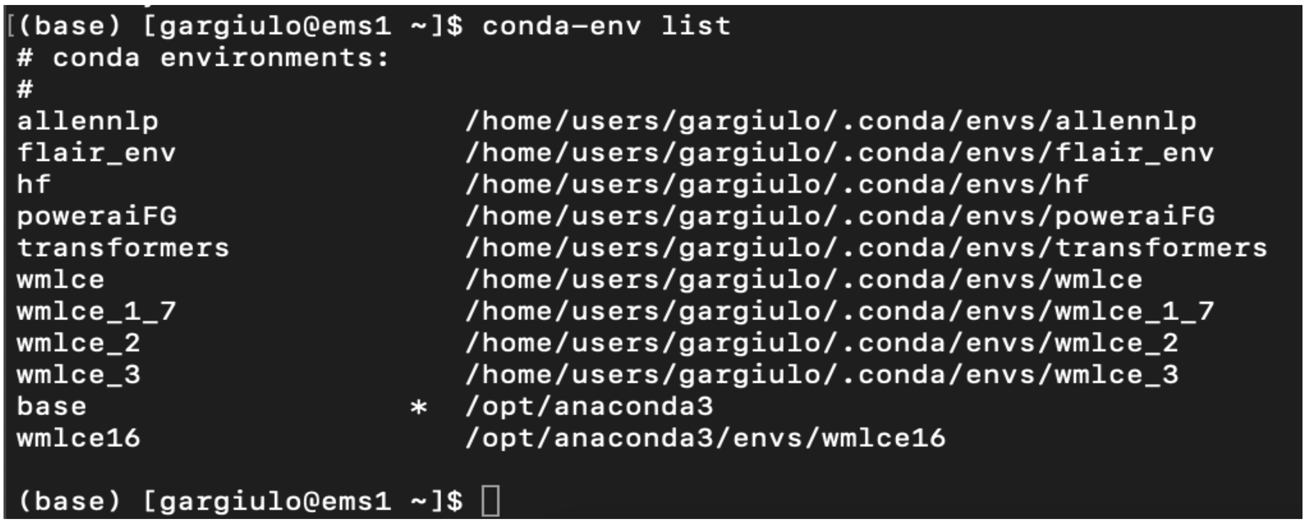
For example, to create an environment named `test_env` with Python version 3.10, you would run:

```
conda create --name test_env python=3.10
conda activate test_env
```

Conda environments are private collections of software that reside in the user's home directory. The directory that contains each user's default personal environment is `~/.conda/envs`.

To immediately access executable tools on the cluster under consideration, below are some relevant cases of interest and useful commands.

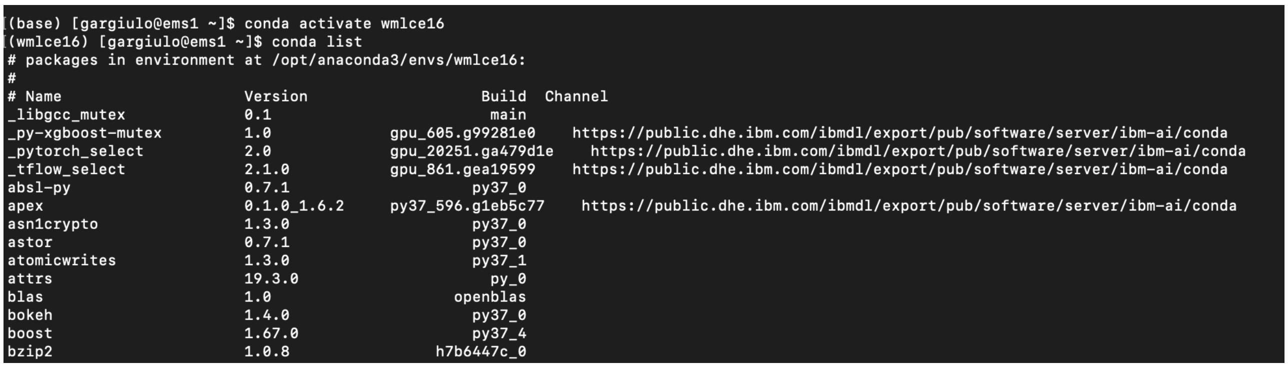
To determine which environments are immediately available, simply execute the command:



```
(base) [gargiulo@ems1 ~]$ conda-env list
# conda environments:
#
allennlp                 /home/users/gargiulo/.conda/envs/allennlp
flair_env                /home/users/gargiulo/.conda/envs/flair_env
hf                       /home/users/gargiulo/.conda/envs/hf
poweraiFG                /home/users/gargiulo/.conda/envs/poweraiFG
transformers             /home/users/gargiulo/.conda/envs/transformers
wmlce                    /home/users/gargiulo/.conda/envs/wmlce
wmlce_1_7                /home/users/gargiulo/.conda/envs/wmlce_1_7
wmlce_2                  /home/users/gargiulo/.conda/envs/wmlce_2
wmlce_3                  /home/users/gargiulo/.conda/envs/wmlce_3
base                  *  /opt/anaconda3
wmlce16                  /opt/anaconda3/envs/wmlce16

(base) [gargiulo@ems1 ~]$ 
```

In this case, you can observe environments associated only with the user "gargiulo," while two environments are valid for all users: base and wmlce16.

Now, you can activate one of the environments (for example, wmlce16) and check which packages are available in it using the commands:



```
(base) [gargiulo@ems1 ~]$ conda activate wmlce16
(wmlce16) [gargiulo@ems1 ~]$ conda list
# packages in environment at /opt/anaconda3/envs/wmlce16:
#
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                        main
_py-xgboost-mutex         1.0               gpu_605.g99281e0    https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda
_pytorch_select           2.0               gpu_20251.ga479d1e    https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda
_tflow_select             2.1.0             gpu_861.gea19599    https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda
absl-py                   0.7.1                      py37_0
apex                      0.1.0_1.6.2       py37_596.g1eb5c77    https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda
asn1crypto                1.3.0                      py37_0
astor                     0.7.1                      py37_0
atomicwrites              1.3.0                      py37_1
attrs                     19.3.0                       py_0
blas                      1.0                      openblas
bokeh                     1.4.0                      py37_0
boost                     1.67.0                     py37_4
bzip2                     1.0.8                h7b6447c_0
```

Using the second command (**conda list**), you can list all the packages in the active environment, specifying, for each one, the package name, version, version details (build), and the channel from which the package was downloaded. Additionally, once the environment is activated, the initial text in parentheses changes, indicating to the user which environment is active, from **(base)** to **(wmlce16)**.

Listing and Activating Environments

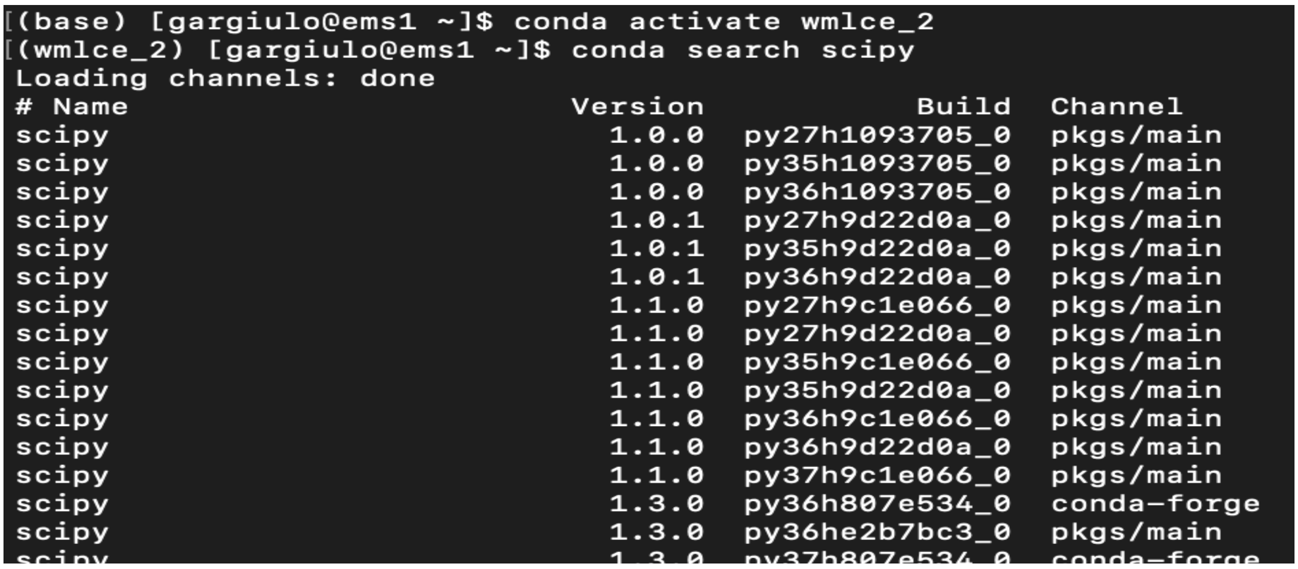
To deactivate the environment, simply execute the command:

```
conda deactivate
```

To install, update, or delete packages, it is best to clone the desired environment into your own using the following command:

```
conda create --name myclone --clone myenv
```

Now, let's see how to install a package. Suppose you need to install "`scipy`".



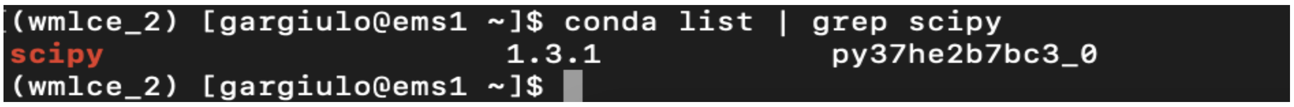
```
[(base) [gargiulo@ems1 ~]$ conda activate wmlce_2
[(wmlce_2) [gargiulo@ems1 ~]$ conda search scipy
 Loading channels: done
 # Name                       Version           Build  Channel
 scipy                          1.0.0  py27h1093705_0  pkgs/main
 scipy                          1.0.0  py35h1093705_0  pkgs/main
 scipy                          1.0.0  py36h1093705_0  pkgs/main
 scipy                          1.0.1  py27h9d22d0a_0  pkgs/main
 scipy                          1.0.1  py35h9d22d0a_0  pkgs/main
 scipy                          1.0.1  py36h9d22d0a_0  pkgs/main
 scipy                          1.1.0  py27h9c1e066_0  pkgs/main
 scipy                          1.1.0  py27h9d22d0a_0  pkgs/main
 scipy                          1.1.0  py35h9c1e066_0  pkgs/main
 scipy                          1.1.0  py35h9d22d0a_0  pkgs/main
 scipy                          1.1.0  py36h9c1e066_0  pkgs/main
 scipy                          1.1.0  py36h9d22d0a_0  pkgs/main
 scipy                          1.1.0  py37h9c1e066_0  pkgs/main
 scipy                          1.3.0  py36h807e534_0  conda-forge
 scipy                          1.3.0  py36he2b7bc3_0  pkgs/main
 scipy                          1.3.0  py37h807e534_0  conda-forge
```

First of all, it is necessary to activate a local user environment. In this case, the user environment **wmlce_2** has been chosen. Then, you search for the package and proceed to install the desired version.

Currently, on IBM channels, the most updated version is 1.5.3, available on **conda-forge**:

```
scipy  1.5.3  py39h6d1a728_0  conda-forge
```

In the example environment **wmlce_2**, a not-so-up-to-date version of `scipy` is currently installed, as seen in the following screenshot:



```
[(wmlce_2) [gargiulo@ems1 ~]$ conda list | grep scipy
 scipy                     1.3.1            py37he2b7bc3_0
 (wmlce_2) [gargiulo@ems1 ~]$
```

If you wanted to install a specific version of the package, you would need to execute the command:

```
conda install scipy=1.5.3
```

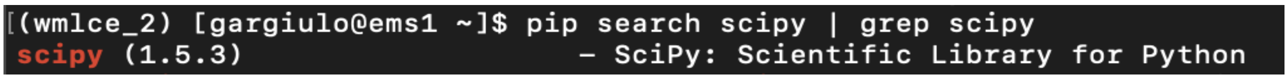
Instead, to update it, you need to execute the command:

```
conda update scipy
```

In case you need to remove it, simply execute the command:

```
conda remove scipy
```

In a conda environment, installing packages using Python's pip command is also possible. Pip (short for Pip Installs Packages) is a command-line tool that allows you to install packages similarly to conda. The packages are sourced from PyPI (Python Package Index), a global repository of thousands of Python-related projects and programs. These are managed and organized based on package versions and their dependencies. Refer to the documentation for a comprehensive list of all the options available for this command.



```
[(wmlce_2) [gargiulo@ems1 ~]$ pip search scipy | grep scipy
scipy (1.5.3)                      - SciPy: Scientific Library for Python
```

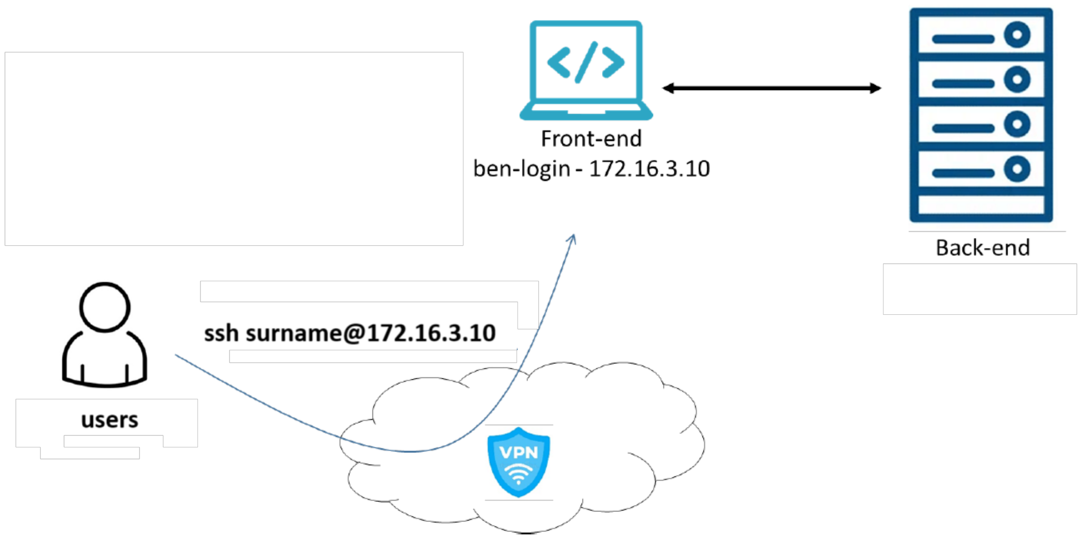
In this case, the most up-to-date version of **`scipy`** on the conda channel matches the latest version on PyPI. However, this is not always the case, as the latest compiled versions for the POWERPC platform are not always available on this channel.

4. Cluster Access Modes

The cluster features two front-end nodes for redundancy: "**ben-login**" and "**ems1**". Users can connect to either of these through a VPN. The internal addresses are 172.16.3.9 for ems1 and 172.16.3.10 for ben-login. Authentication occurs via SSH keys. Using a VPN ensures a secure connection to the cluster.

The following Figure illustrates the main steps that a user must follow to connect to the IBM cluster through the ben-login front-end:

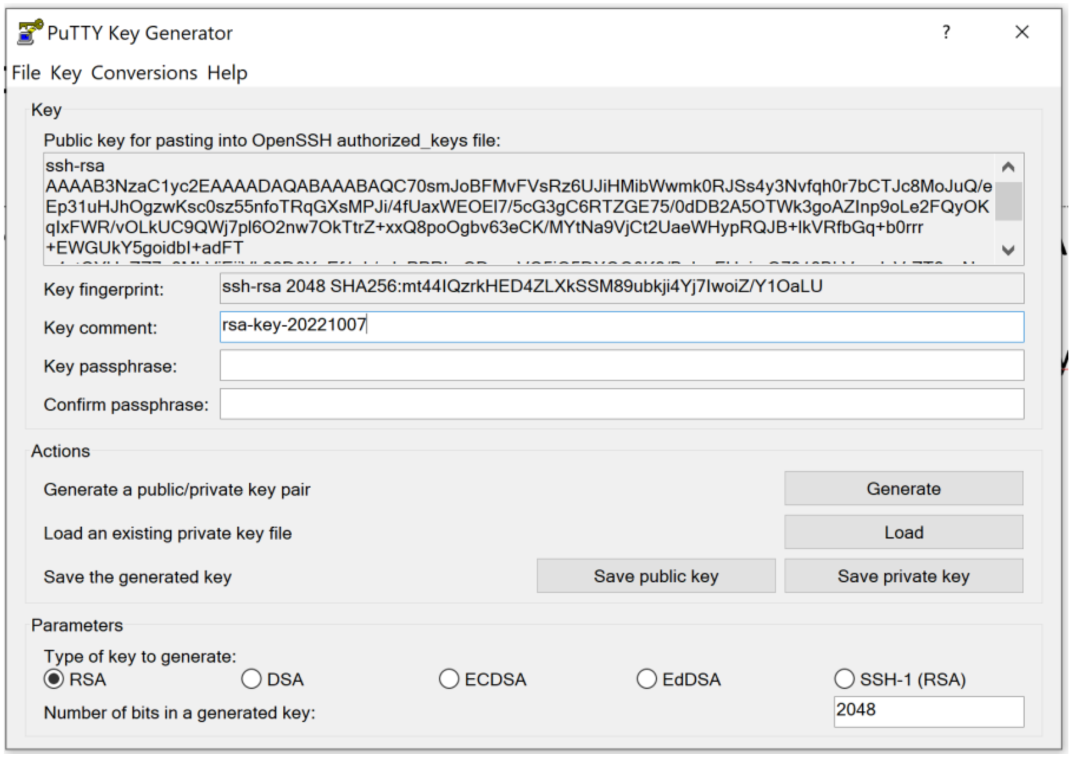
1. Activation of the VPN network (OpenVPN)
2. Verification of front-end reachability
3. SSH authentication



Before accessing the cluster, users must meet the following requirements:

- A VPN certificate for connecting to the front end (provided by the system administrator)
- A pair of RSA keys (public and private)

For Windows users, it is possible to generate the keys using PuTTYgen as follows:



Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC70smJoBFMvFVsRz6UJiHMibWwmk0RJSs4y3Nvfqh0r7bCTJc8MoJuQ/e
Ep31uHJhOgzwKsc0sz55nfoTRqGXsMPJi/4fUaxWEOEI7/5cG3gC6RTZGE75/0dDB2A5OTWk3goAZInp9oLe2FQyOK
qIxFWR/vOLkUC9QWj7pl6O2nw7OkTtrZ+xxQ8poOgbv63eCK/MYtNa9VjCt2UaeWHypRQJB+IkVRfbGq+b0rrr
+EWGUkY5goidbI+adFT
```

Key fingerprint: ssh-rsa 2048 SHA256:mt44IQzrkHED4ZLXkSSM89ubkji4Yj7IwoiZ/Y1OaLU

Key comment: rsa-key-20221007

Key passphrase:

Confirm passphrase:

Generate a public/private key pair

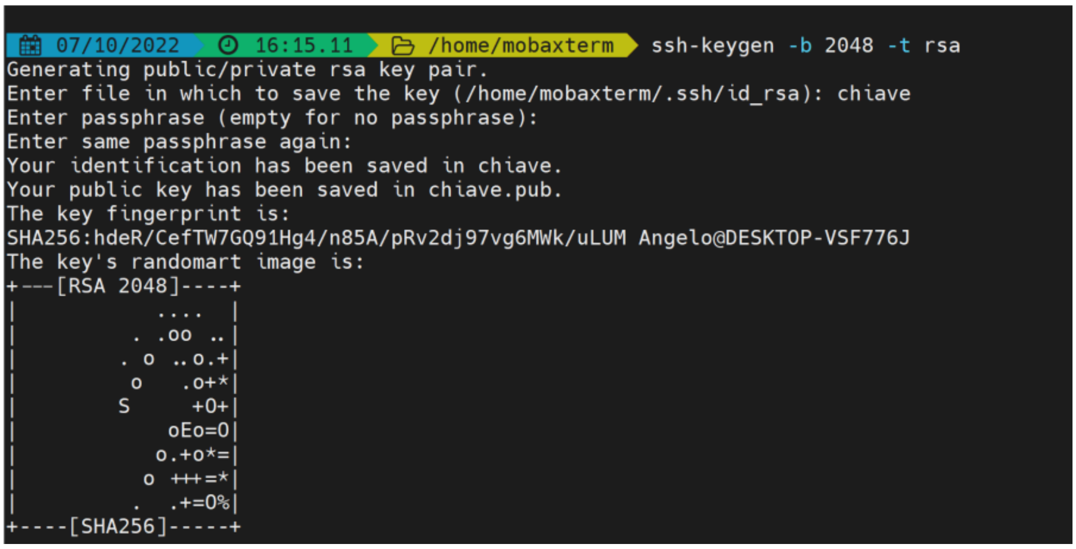
Load an existing private key file

Save the generated key

Type of key to generate:
[✓ RSA] DSA ECDSA EdDSA SSH-1 (RSA)

Number of bits in a generated key: 2048

For Linux users, the following shell command can be used: ssh-keygen -b 2048 -t rsa

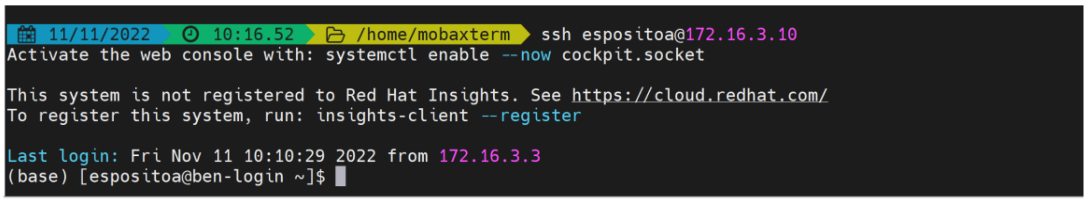


```
 07/10/2022   16:15.11   /home/mobaxterm  ssh-keygen -b 2048 -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mobaxterm/.ssh/id_rsa): chiave
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in chiave.
Your public key has been saved in chiave.pub.
The key fingerprint is:
SHA256:hdeR/CefTW7GQ91Hg4/n85A/pRv2dj97vg6MWk/uLUM Angelo@DESKTOP-VSF776J
The key's randomart image is:
+---[RSA 2048]----+
|           ....  |
|         . .oo ..|
|        . o ..o.+|
|         o   .o+*|
|        S     +0+|
|            oEo=0|
|           o.+o*=|
|          o +++=*|
|         .   .+=0%|
+----[SHA256]-----+
```

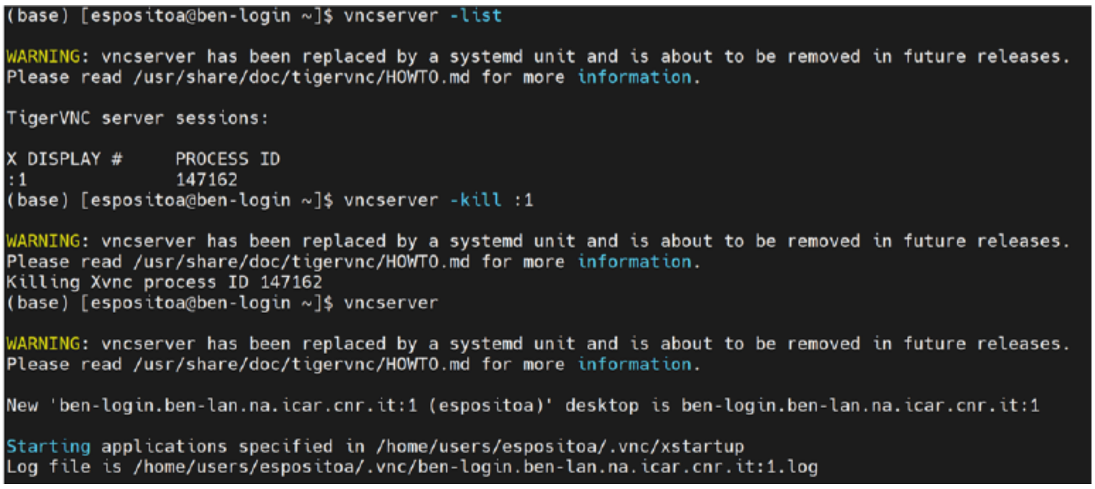
The public key must be provided to the Cluster administrator while the user uses the private key for authentication.

Once authenticated, it is possible to launch, if necessary, a graphical interface using vncserver[4].

The following figures show the SSH connection to the front end and the subsequent launch of a vncserver graphical session.



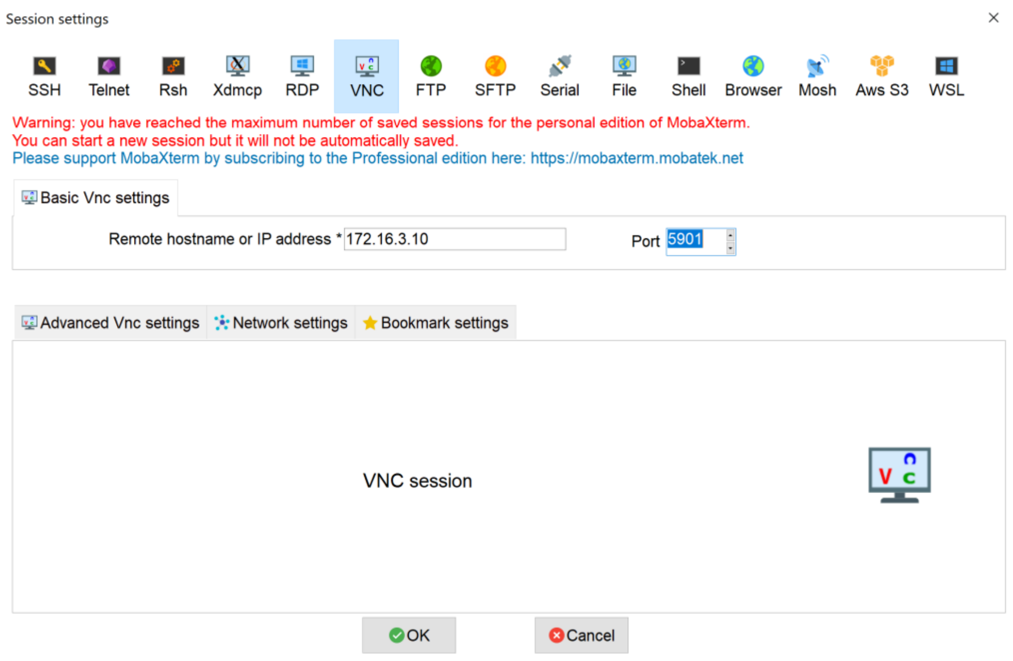
```
 11/11/2022   10:16.52   /home/mobaxterm  ssh espositoa@172.16.3.10
Activate the web console with: systemctl enable --now cockpit.socket

This system is not registered to Red Hat Insights. See https://cloud.redhat.com/
To register this system, run: insights-client --register

Last login: Fri Nov 11 10:10:29 2022 from 172.16.3.3
(base) [espositoa@ben-login ~]$ 
```



```
(base) [espositoa@ben-login ~]$ vncserver -list

WARNING: vncserver has been replaced by a systemd unit and is about to be removed in future releases.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.

TigerVNC server sessions:

X DISPLAY #     PROCESS ID
:1              147162
(base) [espositoa@ben-login ~]$ vncserver -kill :1

WARNING: vncserver has been replaced by a systemd unit and is about to be removed in future releases.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.
Killing Xvnc process ID 147162
(base) [espositoa@ben-login ~]$ vncserver

WARNING: vncserver has been replaced by a systemd unit and is about to be removed in future releases.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.

New 'ben-login.ben-lan.na.icar.cnr.it:1 (espositoa)' desktop is ben-login.ben-lan.na.icar.cnr.it:1

Starting applications specified in /home/users/espositoa/.vnc/xstartup
Log file is /home/users/espositoa/.vnc/ben-login.ben-lan.na.icar.cnr.it:1.log
```

[4] https://www.realvnc.com/

To create a new VNC connection, as illustrated in the figure below, it is sufficient to add the connection port and the address of the front-end server to activate the graphical session.



Session settings

SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL

Warning: you have reached the maximum number of saved sessions for the personal edition of MobaXterm.
You can start a new session but it will not be automatically saved.
Please support MobaXterm by subscribing to the Professional edition here: https://mobaxterm.mobatek.net

Basic Vnc settings

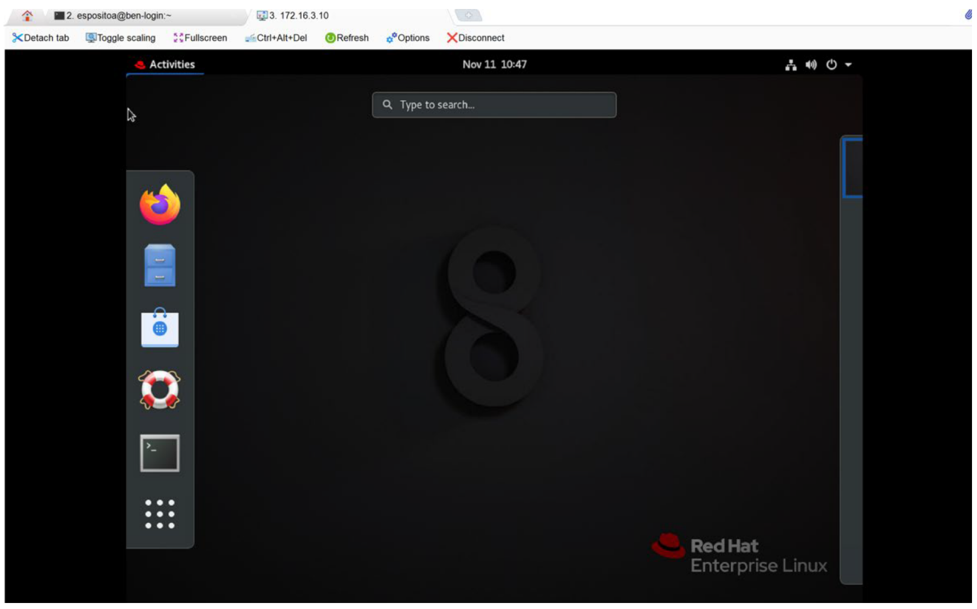
Remote hostname or IP address * 172.16.3.10

Port 5901

Advanced Vnc settings | Network settings | Bookmark settings

VNC session

OK | Cancel



5. Conclusion

This technical report presented BEN, an IBM-produced high-performance computing cluster acquired by the Institute of High-Performance Computing and Networks (ICAR) of the National Research Council (CNR) and other CNR Research Institutes.

This operational guide aimed to provide a comprehensive overview of the BEN-Cluster, outlining its hardware specifications, job management through the Platform Load Sharing Facility (LSF), and software environment setup and management using Conda. Detailed instructions on simulating a workstation environment via VNC are also included, facilitating a seamless transition for researchers from their local machines to the cluster environment.

Future improvements will further align the infrastructure with GDPR directives and optimize service management. Several operational modes for using this platform have been introduced. In the future, if usage scenarios and requirements change, it might be possible to add or radically modify how users interact with the cluster. For example, the modeling part should be optimized by providing a front-end based on Jupyter/Anaconda.

References

1. **IBM LSF Documentation** - Comprehensive guide on using LSF for job scheduling and resource management. Available at: IBM LSF Documentation
2. **LSF Commands Quick Reference** - A quick reference guide for commonly used LSF commands. Available at: LSF Commands Quick Reference

3. Conda Documentation - Comprehensive guide on using Conda for environment and package management.
4. IBM Power Systems Technical White Paper - A comprehensive guide on the POWER9 architecture and its applications in high-performance computing.
5. NVIDIA Tesla V100 GPU Overview - Detailed specifications and performance metrics for the Tesla V100 GPU.
6. HPC with IBM POWER9 and NVIDIA GPUs - A study on the integration of IBM POWER9 processors with NVIDIA GPUs in HPC environments.