Central Computing Facility
Indian Institute of Information Technology, Allahabad

Big Data / Cloud / HPC / VM Usage Manual

The CCF Committee manages the IIITA Big Data / Cloud / HPC / VM System. The Committee developed the User Policies here. The user policies below aim to help make the system effective for users, ensure proper maintenance of the system, and enable researchers to contribute to the system.
User Policies are continually reviewed and updated by the Committee. Questions or concerns should be e-mailed to fi.ccf@iiita.ac.in.

It is important that all users have access to the CCF system and that the integrity of all users' code will maintained. Following the CCF usage policies would require that users understand the nature of computing and the configuration of the hardware and software on the system. Please contact fi.ccf@iiita.ac.in to ask questions or to report problems. If a user does not abide by the CCF usage Policies, the CCF Committee will have the right to take disciplinary action against the user's account.

Users must use the batch submission system of the scheduler:
Users must login to the head node and use the scheduler to submit their jobs on the HPC cluster. Users must not log into the compute nodes for running a job directly without the scheduler.

Correspondence from the CCF Committee:
Users will be notified by e-mail about issues related to the system, such as scheduled downtime, upgrades, etc. Requests for information and feedback about system use and access will be made by e-mail.

Obtaining an account:
To get an account on Big Data / Cloud / HPC / VM System user has to fill a form available at CCF, 5521 Computer Centre-3, and have to give a proper justification to use the Big Data / Cloud / HPC / VM System. If their justification is valid, they will be given an account on CCF Machines.

Preference will be given to Researchers/Students/Faculty of IIITA. Any collaborator/student not belonging to IIITA has to get an authorization from the host in IIITA and the discretion to approve it, will lie entirely with the CCF Committee.

Accessing a user account:
A user having an account on Big Data / Cloud / HPC / VM System can access the cluster with the command:

- For HPC : ssh <username>@172.20.70.12
- For BigData : ssh <username>@172.20.70.16
- For Cloud : ssh <username>@IPAddress
- For VM : ssh <username>@IPAddress

Users are strictly advised against sharing their access details with other users.
The scheduler reserves available compute nodes and other resources on a firstcome-first-served basis among Users with equal priority, but this do not apply to system administration and testing of the machine.

Computing resource usage policy:

The Big Data Facility "DHYAN" has 20 data nodes having 800 Core and 8.0TB of RAM.

- The Big Data Facility having these software components: Ambari, Atlas, Flume, HBase ecosystem, Hive, Kafka, Mahout, Map Reduce, Oozie, PIG, Slider, Spark, Spark2, Tez, YARN, Zookeeper.

The Cloud Facility has 6 cloud nodes having 240 cores and 2.3TB of RAM.

- The Cloud Facility supports 10Gbps internal / external connectivity, Virtual Machine and LXC, LXD, Docker containers and IaaS, CaaS, SaaS.

The HPC Cluster "SURYA" has one master node and 20 compute nodes having 800 Cores and 8.5TB of RAM.

- Each 16-compute node having 40 processors and 370 GB of DDR4 RAM.
- Each 4-compute node having two NVIDIA TESLA V-100 (16GB) GPGPU.
- Access to HPC cluster is to be via secure communication channel (e.g. ssh) to the master node. Compute nodes are intended for handling heavy computational work, and must be accessed via the resource management system (Torque) only.

- Each individual user will assigned a standard storage allocation or quota on /home. Each user on SURYA has a default quota of 10GB on the home directory.
- Users who utilize more than their allocated space may not be able to submit jobs from their home directory until they clean their space and reduce their usage, or they can also request for additional storage with proper justification, which may allocated to user, subjected to the availability of space.
- All jobs submitted to cluster via the resource management system (queue). This enables resources to be sensibly allocated and most efficiently used.
- There is no system backup for data in /home or any other partition, it is the user's responsibility to back up his/her data on a regular basis that is not stored in his/her home directory. We cannot recover any data in any of location, including files lost to system crashes or hardware failure so it is important to make copies of your important data regularly.
- Users must report for any weaknesses in computer security and incidents of possible misuse or violation of the account policies to the HPC administrators or write regarding the same to fi.ccf@iiita.ac.in.
- Any processes that may create performance, load issues on the master node, or interfere with other users' jobs will be terminated.

SUBMITTING JOB ON BIG DATA CLUSTER:

- *Apache Spark*

  Spark built on the concept of Distributed Datasets, which contain arbitrary Java or Python objects. You create a dataset from external data, and then apply parallel operations to it. The building block of the Spark API is its RDD API. In the RDD API, there are two types of operations: transformations, which define a new dataset based on previous ones, and actions, which kick off a job to execute on a cluster. On top of Spark's RDD API, high-level APIs are provided, e.g. DataFrame API and Machine Learning API. These high-level APIs provide a concise way to conduct certain data operations. In this page, we will show examples using RDD API as well as examples using high level APIs.

  Note: Your files or directory path defined in hdfs path:
  Check HDFS file system: hdfs dfs –ls /user/username
  For more info: hdfs dfs -help

  Spark examples:

spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --driver-memory 4g --executor-memory 2g --executor-cores 1 --queue default spark-examples_2.11-2.3.0.2.6.5.0-292.jar 10

sperk-submit: submitting job to spark cluster via yarn master to cluster
spark executable jar: spark-examples_2.11-2.3.0.2.6.5.0-292.jar *arguments:* 10
HDFS example for Spark (All exaples are written in python)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
       .map(lambda word: (word, 1)) \
       .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

- *Data Frame API*

  In Spark, a DataFrame is a distributed collection of data organized into named columns. Users can use DataFrame API to perform various relational operations on both external data sources and Spark's built-in distributed collections without providing specific procedures for processing data. Also, programs based on DataFrame API will be automatically optimized by Spark's built-in optimizer, Catalyst. Examples:

  Text Search: In this example, we search through the error messages in a log file. (Written in python)

```
textFile = sc.textFile("hdfs://...")
# Creates a DataFrame having a single column named "line"
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))
# Counts all the errors
errors.count()
# Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count()
# Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect()
```

  Simple Data Operations: In this example, we read a table stored in a database and calculate the number of people for every age. Finally, we save the calculated result to S3 in the format of JSON. A simple MySQL table "people" is used in the example and this table has two columns, "name" and "age". Examples: (Written in python)

```
# Creates a DataFrame based on a table named "people"
# stored in a MySQL database.
url = \

"jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassw
ord"
df = sqlContext \
```

```
.read \
.format("jdbc") \
.option("url", url) \
.option("dbtable", "people") \
.load()

# Looks the schema of this DataFrame.
df.printSchema()

# Counts people by age
countsByAge = df.groupBy("age").count()
countsByAge.show()

# Saves countsByAge to S3 in the JSON format.
countsByAge.write.format("json").save("s3a://...")
```

- *Machine learning Example*

  MLlib, Spark's Machine Learning (ML) library, provides many distributed ML algorithms. These algorithms cover tasks such as feature extraction, classification, regression, clustering, recommendation, and more. MLlib also provides tools such as ML Pipelines for building workflows, CrossValidator for tuning parameters, and model persistence for saving and loading models.

  Prediction with Logistic Regression: **In this example, we take a dataset of labels and feature vectors. We learn to predict the labels from feature vectors using the Logistic Regression algorithm. Examples:**

```
# Every record of this DataFrame contains the label and
# features represented by a vector.
df = sqlContext.createDataFrame(data, ["label", "features"])

# Set parameters for the algorithm.
# Here, we limit the number of iterations to 10.
lr = LogisticRegression(maxIter=10)

# Fit the model to the data.
model = lr.fit(df)

# Given a dataset, predict each point's label, and show the results.
model.transform(df).show()
```

Spark comes with several sample programs. Scala, Java, Python and R examples are in the examples/src/main directory. To run one of the Java or Scala sample programs, use bin/run-example <class> [params] in the top-level Spark directory. (Behind the scenes, this invokes the more general spark-submit script for launching applications). Example are available at given path: Directory location: /usr/hdp/2.6.5.0-292/spark/bin/run-example

SUBMITTING QUEUE ON HPC CLUSTER:

- Use job submission script and "qsub" command to submit your job using scheduler. (Kindly don't use #PBS -d <directory name> option in the PBS script, instead use "cd <directory name>"
- You can check cluster module availability on the Head Node with the command: "module avail"
- Load the module: "module load <module_name>"
- Available queues in the cluster
    - core160 : 360 Hrs with 160 Cores.
    - core320 : 24 Hrs 320 Cores.
    - gpu : 360 Hrs with 20/40 Cores and 1/2 Tesla V-100 GPU.
- Use the command "qstat -a" to see the status of your jobs.
- Use the command "qstat -n" to see which nodes your job is running.
- Use the command "qstat -f <jobid>" to see detail information about your submitted job.
- Use "qdel <job id>" to kill your job.
- To force fully kill your job use the command "qdel -Wforce <jobid>.

Sample job script are given

***Core160* Queue:**

```
#!/bin/bash

#PBS -u USER_NAME
#PBS -N STUDENT_NAME
#PBS -q core160
#PBS -l nodes=4:ppn=40
#PBS -o out.log
#PBS -e error.log

module load compilers/intel/parallel_studio_xe_2018_update3_cluster_edition

cd $PBS_O_WORKDIR

mpiexec.hydra -f $PBS_NODEFILE -np 160 "/SCRIPT_PATH/"

./Job_script.sh
exit;
```

***Core320* Queue:**

```
#!/bin/bash

#PBS -u USER_NAME
#PBS -N STUDENT_NAME
#PBS -q core320
#PBS -l nodes=8:ppn=40
#PBS -o out.log
#PBS -e error.log

Module load compilers/intel/parallel_studio_xe_2018_update3_cluster_edition

cd $PBS_O_WORKDIR

mpiexec.hydra -f $PBS_NODEFILE -np 320 "/SCRIPT_PATH/"

./Job_script.sh
exit;
```

***GPU* Queue:**

```
#!/bin/bash

#PBS -u USER_NAME
#PBS -N STUDENT_NAME
#PBS -q gpu
#PBS -l select=1:ncpus=20:ngpus=1 (For ONE GPU)
#PBS -l select=2:ncpus=20:ngpus=1 (For TWO GPU)
#PBS -o out.log
#PBS -e error.log

Module load compilers/intel/parallel_studio_xe_2018_update3_cluster_edition

export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH

cd $PBS_O_WORKDIR

python your_script_name.py

mpiexec.hydra -np 2 your script_name.sh

exit;
```

Note:

- Every user will allowed running only a single job on HPC Cluster. However, the jobs already submitted will continue to run unless it gets over.
- No two or more jobs you can submit from the same user account.
- If all the nodes are running and you are submitting your first single job, your job will get a queue. Once the nodes are free, your job will start running automatically.
- Please note that, the support team may delete multiple jobs submitted by single user without prior notice to user.

Users must:

Acknowledge the use of the HPC system in papers and presentations. A recommended acknowledgement is "The computational results reported in this work were performed on the Central Computing Facility of IIITA, Allahabad ".

Policy violations:

If it has been determined that any user has violated any of the HPC resource policies, or any other computing policies, then the user will be liable for strict disciplinary action by the CCF Committee.

Getting help:

If you have any questions or concerns regarding any of these policies, please send an email to fi.ccf@iiita.ac.in or get in touch here.

Central Computing Facility
C. V. Raman Bhavan (Computer Centre -3),
Fifth Floor, Room No.-5521,
Indian Institute of Information Technology-Allahabad,
Prayagraj-211015.
Ext. No. – 0532-292-2523.