Turing Cluster

Note : Under Construction. Suggestions Welcome

Master Node: turing.cds.iisc.ac.in [10.24.1.32]
Worker Nodes: node1 - node24 [10.24.1.201 - 10.24.1.224]

To Login to master node:
How to access the turing cluster :

ssh <username>@turing.cds.iisc.ac.in

To have a passwordless login with ssh, copy your ssh keys to turing master node. Once you copy your keys, you can login without password next time using ssh.

ssh-copy-id <username>@turing.cds.iisc.ac.in

To have a X window environment (to access GUI programs),

ssh -X <username>@turing.cds.iisc.ac.in and run your program.

To Login to worker nodes: (rarely required)

ssh <username>@node with same password as the master node.

Getting Acquainted :
When you login, you'll be in your home directory (/home/<username>). You can use this directory to install user specific libraries, tools and also your code. **Since we have a limited capacity on /home please don't store any large datasets here.** You'll be allotted space on /scratch partition later when you need to store larger datasets.

Available tools and languages :

Screen Managers : If you need to run your programs even after logging out or have your sessions not terminated, please use **screen** or **tmux** both of which are available on master node.

Editors : Vim, Sublime and Emacs are installed systemwide. Please install any specific editors or IDEs (eclipse, netbeans, intelliJ or pycharm etc) on your home folders.

Version Control : Git

Languages :
Python -

Python 2 - python or python2.7 (to install python 2 packages locally on your home folder please use a virtual env or install on your home folder with ***pip install <package> --user***)

Python 3 - python3 (to install python 3 packages, use pip3. By default it'll install on your home folder)
Java : Java 8 is installed and available system wide.

Hadoop Framework:
From Hadoop website -
The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets

The Turing cluster runs Hadoop 2.7.3 release. You can the dashboard available at http://turing.cds.iisc.ac.in:8080 with credentials username - ***turing-hadop-user*** and passwrd - ***turing123*** to see read-only configuration of Hadoop cluster.

The main components you have to be familiar are HDFS and Yarn Scheduler. You'll use HDFS to store your data and Yarn to schedule and run your jobs on the Hadoop cluster.

HDFS :
HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. The HDFS Architecture Guide describes HDFS in detail. This user guide primarily deals with the interaction of users and administrators with HDFS clusters. The HDFS architecture diagram depicts basic interactions among NameNode, the DataNodes, and the clients. Clients contact NameNode for file metadata or file modifications and perform actual file I/O directly with the DataNodes.

HDFS provides commands analogous to a POSIX filesystem for interacting with. Please see here : https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html for brief overview of HDFS filesystem commands.

Yarn : Yarn is the job scheduler and resource manager for Hadoop. A good overview is availble at https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/YARN.html.

Running your job on Hadoop :
First, create a jar of your implemented code. Once you have your jar, you can run it with the following command:

hadoop jar name-of-jar.jar <options> path/to/input path/to/output

Note that the path to input and output is on HDFS, not the local machine. This will read input from HDFS and store output on HDFS

Hadoop Streaming:
You can use any executables as mappers and reducers (written in languages other than java like pyhton, shell or any other executable). Please see the documentation on the hadoop site - https://hadoop.apache.org/docs/r1.2.1/streaming.html for an introduction.

The location of the streaming JAR for the installation on Turing is - /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar

Monitoring your job progress : You can use resource manager web page available at - http://turing.cds.iisc.ac.in:8088/cluster to monitor your job progress.

To seamlessly work with dashboard from your desktop or laptop, you can add these entries to your /etc/hosts file

```
10.24.1.32    turing.cds.iisc.ac.in    turing
10.24.1.201    node1.local    node1    n1
10.24.1.202    node2.local    node2    n2
10.24.1.203    node3.local    node3    n3
10.24.1.204    node4.local    node4    n4
10.24.1.205    node5.local    node5    n5
10.24.1.206    node6.local    node6    n6
10.24.1.207    node7.local    node7    n7
10.24.1.208    node8.local    node8    n8
10.24.1.209    node9.local    node9    n9
10.24.1.210    node10.local    node10    n10
10.24.1.211    node11.local    node11    n11
10.24.1.212    node12.local    node12    n12
10.24.1.213    node13.local    node13    n13
10.24.1.214    node14.local    node14    n14
10.24.1.215    node15.local    node15    n15
10.24.1.216    node16.local    node16    n16
```

10.24.1.217	node17.local	node17	n17
10.24.1.218	node18.local	node18	n18
10.24.1.219	node19.local	node19	n19
10.24.1.220	node20.local	node20	n20
10.24.1.221	node21.local	node21	n21
10.24.1.222	node22.local	node22	n22
10.24.1.223	node23.local	node23	n23
10.24.1.224	node24.local	node24	n24

Apache Spark on Turing Cluster :

From Spark website (https://spark.apache.org/) :

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, **MLlib for machine learning, GraphX for graph processing**, and Spark Streaming. Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

On Turing cluster, we will be using Spark running on Hadoop (using YARN as resource manager and HDFS as file system). If you need Spark in different configuration, please let us know.

There are two versions of Spark installed on turing (Spark 1 and Spark 2). You can use spark with either scala or python. You can run Spark either in interactive REPL mode or self-contained application mode.

Running Spark Interactively :

Using Spark with Scala :
To use version 1 run : ***/usr/hdp/current/spark-client/bin/spark-shell*** or you can also define SPARK_MAJOR_VERSION=1 by running ***export SPARK_MAJOR_VERSION=1*** and simply run ***spark-shell***

To use version 2 run : ***/usr/hdp/current/spark2-client/bin/spark-shell*** or you can also define SPARK_MAJOR_VERSION=2 by running ***export SPARK_MAJOR_VERSION=2*** and simply run ***spark-shell***

Using Spark with Python :
To use version 1 run : ***/usr/hdp/current/spark-client/bin/pyspark*** or you can also define SPARK_MAJOR_VERSION=1 by running ***export SPARK_MAJOR_VERSION=1*** and simply run ***pyspark***

To use version 2 run : ***/usr/hdp/current/spark2-client/bin/pyspark*** or you can also define SPARK_MAJOR_VERSION=2 by running ***export SPARK_MAJOR_VERSION=2*** and simply run ***pyspark***

You can check here - https://spark.apache.org/docs/latest/quick-start.html#interactive-analysis-with-the-spark-shell for a quick start on both python and scala interactive shell.

Self-Contained Applications :

Self-contained applications provide a batch mode to run the applications written using the Spark API. You can use Scala or Python or Java to write applications with Spark API and run them on cluster mode with YARN scheduler.

You can see https://spark.apache.org/docs/latest/quick-start.html#self-contained-applications for a quick start on self-contained applications.

Submitting to YARN scheduler on turing cluster :

We use ***spark-submit*** to submit applications to YAR scheduler. Similar to interactive shell, you can define SPARK_MAJOR_VERSION or use full path and use corresponding version of spark to run your jobs.

For a quick test spark-submit, you can run the command below to submit an app to YARN scheduler and see it gets scheduled on one or more nodes of HAdoop cluster.

spark-submit --master yarn /usr/hdp/current/spark2-client/examples/src/main/python/pi.py 100

You should be able to see it get scheduled by YARN on one of the nodes and you can check for its progress using resource manager UI available at http://turing.cds.iisc.ac.in:8088/cluster

Similarly you can schedule your self-contained apps written in java or scala by builing jars and submitting to YARN using spark-submit.

More information on cluster scheduling is available at
https://spark.apache.org/docs/latest/running-on-yarn.html

Reading and Writing from HDFS :
The below code fragment shows how you can read from and write to HDFS using Spark API.

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