4. Spark Installation Notes

May 14, 2021

1 Spark Installation

There are two well supported deployment modes for Spark:

- **Local**: Good for working on a local desktop typically with smaller/sampled datasets. Local mode is an excellent way to learn and experiment with Spark. Local mode also provides a convenient development environment for analyses, reports, and applications that you plan to eventually deploy to a multi-node Spark cluster.
- **Cluster**: Good for working directly within or alongside a Spark cluster. The Spark system currently supports several cluster managers, such as **standalone**, a simple cluster manager included with Spark that makes it easy to set up a clusterm, and **YARN**, the resource manager in Hadoop.

We will install Spark 2.4.4 on top of the Hadoop cluster we already installed and configured. In this way, we will be able to deploy Spark applications both in local and in cluster-YARN modes.

In the following we will see how:

- Install Spark on your cluster's machines.
- Start Spark on your cluster.

1.1 Download and configure Spark

1.1.1 Setup

We assume that you followed the previous instructions on how to install and configure a three-node Hadoop cluster to set up your YARN cluster. The master node (HDFS NameNode and YARN ResourceManager) is now called **node-master** and the slave nodes (HDFS DataNode and YARN NodeManager) are called **node1**, **node2**, etc.

NOTE 1: *Run the commands in this guide on the* ***node-master*** *unless otherwise specified.*

NOTE 2: These instruction must be executed as the `hadoop` user (a non-root user). Commands that require elevated privileges are prefixed with `sudo`.

1. Be sure you have a `hadoop` user that can access all cluster nodes with SSH keys without a password.
2. Run `jps` *on each of the nodes* to confirm that HDFS and YARN are running. If they are not, start the services with:

```
start-dfs.sh
start-yarn.sh
```

2. We will install the software under the `/opt` folder. On the **node-master** run the following command to create the folders.

```
sudo mkdir /opt/spark
sudo chown -R hadoop /opt
```

2. On the **node-master** download spark-2.4.4-bin-without-hadoop.tgz in your home folder using the following command:

```
wget -c -O ~/spark.tgz https://archive.apache.org/dist/spark/spark-2.4.4/\
spark-2.4.4-bin-without-hadoop.tgz
```

3. On the **node-master** decompress the Spark package. You can use the following command:

```
tar -xvf spark.tgz --directory=/opt/spark --strip 1
```

To save space, remove the `spark.tgz` file from your folder:

```
rm ~/spark.tgz
```

4. On the **node-master** the `/home/hadoop/.bashrc` file must be edit to include the following (add these line *after* the lines we added for Hadoop):

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

Log out and re-login to your `hadoop` account.

5. Rename the spark default template config file:

```
mv $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf
```

6. Edit `$SPARK_HOME/conf/spark-defaults.conf` and set `spark.master` to `yarn`:

```
spark.master    yarn
```

7. Rename the Spark default environment shell script:

```
mv $SPARK_HOME/conf/spark-env.sh.template $SPARK_HOME/conf/spark-env.sh
```

8. Edit `$SPARK_HOME/conf/spark-env.sh` to include the following

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
export PYSPARK_PYTHON=/usr/bin/python3.6
```

Spark is now ready to interact with your YARN cluster.

1.1.2 Understanding the Spark memory allocation

A Spark job consists of two parts: **executors** that run the actual tasks, and a **driver** that schedules the executors.
Spark jobs can run on YARN in two modes: **cluster mode** and **client mode**.

- **Cluster mode**: everything runs inside the cluster. You can start a job from your laptop and the job will continue running even if you close your computer. In this mode, the Spark driver is encapsulated inside the YARN Application Master.

- **Client mode** the Spark driver runs on a client, such as your laptop. If the client is shut down, the job fails. Spark executors still run on the cluster, and to schedule everything, a small YARN Application Master is created.

Client mode is well suited for interactive jobs, but applications will fail if the client stops. For long running jobs, cluster mode is more appropriate.
Understanding the difference between the two modes is important for choosing an appropriate memory allocation configuration, and to submit jobs as expected.

Allocation of Spark containers to run in YARN containers may fail if memory allocation is not configured properly. For nodes with less than 4 GB RAM, the default configuration is not adequate and may trigger swapping and poor performance, or even the failure of application initialization due to lack of memory.
If the memory requested is above the maximum allowed, YARN will reject creation of the container, and your Spark application won't start.

Get the value of `yarn.scheduler.maximum-allocation-mb` in `$HADOOP_HOME/etc/hadoop/yarn-site.xml`. This is the maximum allowed value, in MB, for a single container. Make sure that values for Spark memory allocation, configured in the following section, are below the maximum. This guide will use a sample value of 1536 for `yarn.scheduler.maximum-allocation-mb`. If your settings are lower, adjust the samples with your configuration.

Driver memory allocation In cluster mode, the Spark driver runs inside YARN application master. The amount of memory allocated to the Spark driver in cluster mode (default value is 1 GB) is configured in `spark-defaults.conf` located at `$SPARK_HOME/conf`:

```
spark.driver.memory    512m
```

Application manager memory allocation In client mode, the Spark driver will not run on the cluster, so the above configuration will have no effect.
A YARN application master still needs to be created to schedule the Spark executor, and you can set its memory requirements (default to 0.5 GB) in `spark-defaults.conf` located at `$SPARK_HOME/conf`:

```
spark.yarn.am.memory    512m
```

Executors memory allocation The Spark executors' memory allocation is calculated based on two parameters in `$SPARK_HOME/conf/spark-defaults.conf`:

- `spark.executor.memory`: sets the base memory used in calculation.
- `spark.yarn.executor.memoryOverhead`: is added to the base memory. It defaults to 7% of base memory, with a minimum of 384 MB.

For example, for a `spark.executor.memory`value of 1 GB , the required memory is 1024 MB + 384 MB = 1408 MB. For 512 MB, the required memory will be 512 MB + 384 MB = 896 MB.
To set executor memory to 512 MB, edit the `$SPARK_HOME/conf/spark-defaults.conf` file and add the following line:

```
spark.executor.memory          512m
```

1.2 Start and test Spark

Applications are submitted with the `spark-submit` command. The Spark installation package contains sample applications, like the parallel calculation of *Pi*, that you can run to practice starting Spark jobs.
To run the sample *Pi* calculation, use the following command:

```
spark-submit --deploy-mode client --class org.apache.spark.examples.SparkPi \
$SPARK_HOME/examples/jars/spark-examples_2.11-2.4.4.jar 10
```

The first parameter, `--deploy-mode`, specifies which mode to use, client or cluster.
At a certain point, in the middle of the logs, you should find the following line:

```
...
Pi is roughly 3.1434511434511437
...
```

1.2.1 Monitoring Spark applications

When you submit a job, the Spark driver automatically starts a web UI on port 4040 that displays information about the application. However, when execution is finished, the web UI is dismissed with the application driver and can no longer be accessed.

Spark provides a **History Server** that collects application logs from HDFS and displays them in a persistent web UI. The following steps will enable log persistence in HDFS:

1. Edit `$SPARK_HOME/conf/spark-defaults.conf` and add the following lines to enable Spark jobs to log in HDFS:

```
spark.eventLog.enabled  true
spark.eventLog.dir      hdfs://hadoop-namenode:9820/spark-logs
```

2. Create the `log` directory in HDFS:

```
hdfs dfs -mkdir /spark-logs
```

3. Configure the History Server related properties in `$SPARK_HOME/conf/spark-defaults.conf`:

```
spark.history.provider            org.apache.spark.deploy.history.FsHistoryProvider
spark.history.fs.logDirectory     hdfs://hadoop-namenode:9820/spark-logs
spark.history.fs.update.interval  10s
spark.history.ui.port             18080
```

You may want to use a different update interval than the default 10s. If you specify a bigger interval, you will have some delay between what you see in the History Server and the real time status of your application. If you use a shorter interval, you will increase I/O on the HDFS.

4. Run the History Server:

```
start-history-server.sh
```

Repeat steps from previous section to start a job with `spark-submit` that will generate some logs in the HDFS. Access the History Server by navigating to `http:/hadoop-namenode:18080` in a web browser.

1.2.2 Run the Spark Shell

The Spark Shell provides a simple way to learn the Spark APIs in Python, as well as a powerful tool to analyze data interactively. We suggest to use the Spark shell in local mode only.

1. Make sure that you have `python3` installed.

```
sudo apt install python3
```

2. Create a `python` alias to the `python3` command:

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 10
```

Now you can use `python` and `python3` commands interchangebly in any shell.

3. Check that everything is correctly installed.

```
python --version
Python 3.6.9
python3 --version
Python 3.6.9
```

3. Install the `pyspark` Python module:

```
pip3 install pyspark
```

4. Start it by running the following in the Spark directory:

```
pyspark
```

You should get something like the following.

```
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/
Using Python version 3.6.9 (default, Nov  7 2019 10:44:02)
SparkSession available as 'spark'.
>>>
```

Use `quit()` or `Ctrl-D` to exit the Spark Shell.

[ ]: