Lab 10 of SP19-BL-ENGR-E599-30563

Lab 10 will introduce the configuration of running Spark on multiple cluster nodes

Goal

- Install and configure Apache Spark with YARN

Deliverables

Submit the computation time with following parameters

- executor number: 24
- cores per executor: 2
- memory per executor: 4
- centroids number: 20
- iteration: 10

Evaluation

Lab participation: credit for 1 point based upon a successful completion of the lab tasks

Configure the Hadoop Service on Multiple Nodes

Before starting to Apache Spark, lets set Hadoop on multiple nodes. You can see the allocations below. Every group can access their teammates nodes.

users	nodes
risnaga - vmagadi	j-022, j-079
xinqwu - jl145	j-035, j-018
sumish - skandag	j-027, j-026
rbapat - abmakash	j-023, j-013
sambavan - vjatakia	j-024, j-011
nishjain - arbansal	j-020, j-014
sahmaini - iarora	j-70, j-016
abhishekb	j-012, j-028
jaikumar - ramgattu - susoni	j-017, j-021, j-042
arpishah - shilsing - karsanc	j-015, j-025, j-019

Before doing anything make sure that you can access to teammate's nodes. If it asks, type `yes`. You can see the example below.

```
[sakkas@j-login1 ~]$ ssh j-005
The authenticity of host 'j-005 (172.16.x.x)' can't be established.
ECDSA key fingerprint is SHA256:zz+bReyZkegxbfEwmnLhzILx72oUG/jxoH7qxlvIdHk.
ECDSA key fingerprint is MD5:cb:aa:x1:f2:e1:x5:5b:a5:f5:x4:dc:b4:d2:x4:21:a5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'j-005,172.16.x.x' (ECDSA) to the list of known hosts.
```

If you have a running Hadoop service, first stop it.

```
cd $HADOOP_HOME/sbin/
./stop-yarn.sh
./stop-dfs.sh
```

you may manually check whether the hadoop daemons are cleaned

```
jps
```

There shall be no other process than the `jps`

Sometimes stop commands don't work. In that case, you can kill processes using `kill -9 process-id` command. You can get the process id using `jps` command. For example:

```
[sakkas@j-029]$ jps
23079 NameNode
23287 DataNode
23500 Jps
[sakkas@j-029]$ kill -9 23079
[sakkas@j-029]$ kill -9 23287
[sakkas@j-029]$ jps
23559 Jps
```

Then, we open the slaves file

```
cd $HADOOP_HOME/etc/hadoop
vim slaves
```

Add the two nodes assigned to your group

```
j-xx1
j-xx2
```

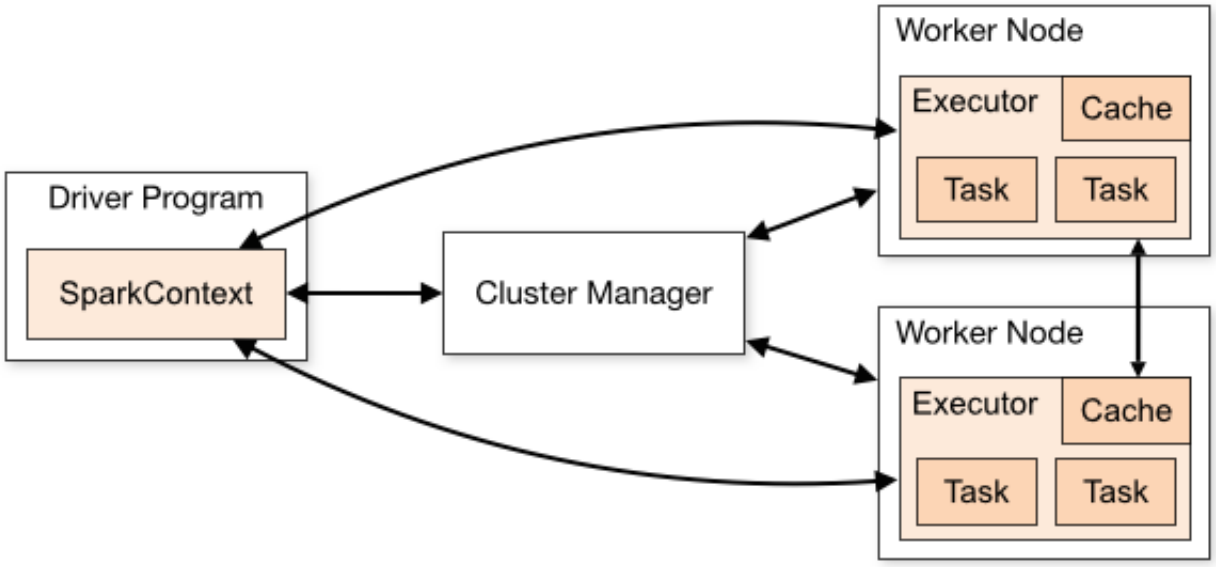
Therefore, we use j-xx1 as the namenode and the first slave node, and j-xx2 as the second slave node.

Note that following part will erase the data inside HDFS. Please backup your data before running it.

```
ssh j-xxx
rm -rf /tmp/hadoop-username
# dont forget to change username with your username
# example: rm -rf /tmp/hadoop-sakkas
hdfs namenode -format
```

Apache Spark Distributed Mode

Apache Spark has three components when running programs on a multiple-node clusters.



Cluster Overview of Apache Spark

Cluster Manager

Cluster manager is in charge of scheduling hardware resources. Apache Spark has its own cluster manager (standalone mode) while supporting other widely used hardware resource manager such as Hadoop YARN.

- Standalone
- Apache Mesos
- Hadoop YARN
- Kubernetes

Driver Program

The driver program contains a `SparkContext` object, which is the main program of Spark and coordinates executor processes and cluster manager. Spark has two deployment modes

- Client Mode: The driver program is running on the local node where it submits the jobs.
- Cluster Mode: The driver program is running within the cluster nodes as other executor processes.

If we consider the local node is near the cluster nodes (named Spark infrastructure) and network latency is not high, the client mode works just fine. However, when the network connection between local node and cluster nodes is slow, it is more stable to run driver program within the cluster nodes.

If the spark is running in an interactive mode (spark-shell), it should use client mode to receive input and return output directly to the users.

Executor Processes

The executor processes are dispatched to the worker nodes (i.e., YARN container). Each executor is in charge of running parallel tasks defined in the program and communicates with the driver program.

Installation

Download the pre-built Apache Spark with support of YARN by using the wget. We choose the version 2.3.2 for Apache Hadoop 2.6 spark-2.3.2-bin-hadoop2.6.tgz

```
user@j-xxx$: cd ~
wget https://archive.apache.org/dist/spark/spark-2.3.2/spark-2.3.2-bin-
hadoop2.6.tgz
tar -xvzf spark-2.3.2-bin-hadoop2.6.tgz
```

Configure your `~/.bashrc` file to add the following environment variables

```
export SPARK_HOME=$HOME/spark-2.3.2-bin-hadoop2.6
export SPARK_CONF_DIR=${SPARK_HOME}/conf
export PATH=${SPARK_HOME}/bin:$PATH
```

Also, make sure that you have your YARN environment variables configured in `~/.bashrc`. It should be already there if you haven't deleted.

```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_CONF_DIR
export YARN_HOME=$HADOOP_HOME
```

Remember to refresh the `~/.bashrc` file

```
source ~/.bashrc
```

Configure Spark with YARN

Rename and edit the `$SPARK_HOME/conf/spark-defaults.conf` file

```
mv ${SPARK_HOME}/conf/spark-defaults.conf.template ${SPARK_HOME}/conf/spark-
defaults.conf
vim ${SPARK_HOME}/conf/spark-defaults.conf
```

Add the lines as follows (don't forget to change username with you user name)

```
spark.master                     yarn
spark.eventLog.enabled           true
spark.eventLog.dir               /scratch_hdd/username/spark/history
spark.history.fs.logDirectory    /scratch_hdd/username/spark/history
spark.driver.memory              20g
```

For `spark.master`, we use `yarn` as the ResourceManager, and please create the evenlog directory under the local disk folder `/scratch_hdd` for each of your Juliet node. Make sure that the spark driver has sufficent memory.

```
# don't forget to change username with your user name
mkdir -p /scratch_hdd/username/spark/history
#to create directory on second node (change xxx with second node and change
username with your user name)
ssh j-xxx "mkdir -p /scratch_hdd/username/spark/history"

#example:
# [sakkas@j-029 ~]$ mkdir -R /scratch_hdd/sakkas/spark/history
# [sakkas@j-029 ~]$ ssh j-005 "mkdir -p /scratch_hdd/sakkas/spark/history"
```

Secondly, edit the YARN configuration file under `$HADOOP_HOME/etc/hadoop/yarn-site.xml`, and add the following lines to disable the memory check, otherwise it may fail the program in yarn-client mode.

```
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
</property>
```

Launch Spark Shell

NOTE THAT ONLY ONE USER CAN RUN HADOOP AND SPARK ON A JULIET NODE. THEREFORE, A STUDENT SHOULD RUN THE FOLLOWING PART FROM EACH GROUP. AFTER THE FIRST STUDENT FINISHES, HE/SHE NEEDS TO STOP HDFS AND YARN. THEN OTHER STUDENT CAN WORK ON THE FOLLOWING PART.

To verify the installation of Spark, we could launch the `spark-shell` program from the local node (your namenode in Hadoop YARN). Before that, make sure that the HDFS and YARN have been launched successfully. Entering the commands at your namenode

```
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh
yarn node -list
```

to get similar screen output

```
Total Nodes:2
         Node-Id             Node-State Node-Http-Address       Number-of-Running-
Containers
j-005.juliet.futuresystems.org:36069            RUNNING j-
005.juliet.futuresystems.org:8042                                  0
j-029.juliet.futuresystems.org:39663            RUNNING j-
029.juliet.futuresystems.org:8042
```

Secondly, use the `spark-shell` command with client deployment mode

```
cd $SPARK_HOME
bin/spark-shell --deploy-mode client
```

to get the following screen output

```
2019-03-22 22:20:51 WARN  NativeCodeLoader:62 - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
2019-03-22 22:20:57 WARN  Client:66 - Neither spark.yarn.jars nor
spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Spark context Web UI available at http://j-029.juliet.futuresystems.org:4040
Spark context available as 'sc' (master = yarn, app id =
application_1553307084028_0003).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.2
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_101)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

We could exit this interactive mode by `Cltr+D`. In the rest of the instructions, we will use the submission mode with Python API for the K-means example.

K-means Example in Spark MLlib

Please copy the folder `pyspark-example` to your home directory

```
cp -a /share/jproject/sakkas/pyspark-example ~/
cd ~/pyspark-example
```

Python codes of K-means

Open the `kmeans.py` file

```
25 from __future__ import print_function
26
27 import sys
28 import time
29 import numpy as np
30 from pyspark.sql import SparkSession
```

The codes use `numpy` and `SparkSession` modules to run the codes. In the main fuction, create a `SparkSession` to launch the spark service.

```
58     spark = SparkSession\
59         .builder\
60         .appName("PythonKMeans")\
61         .getOrCreate()
```

The main body first reads in data and parameters. The input data is stored in the RDD format,

```
63     lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
64     data = lines.map(parseVector).cache()
65     K = int(sys.argv[2])
66     iteration = float(sys.argv[3])
```

where the file is splitted by lines (rows) and a second function `parseVector` to split each line and store it in a numpy array.

```
33 def parseVector(line):
34     return np.array([float(x) for x in line.split(',')])
```

A while loop is then used to compute the centroids distances

```
72  while iteration > 0:
73      closest = data.map(
74          lambda p: (closestPoint(p, kPoints), (p, 1)))
75      pointStats = closest.reduceByKey(
76          lambda p1_c1, p2_c2: (p1_c1[0] + p2_c2[0], p1_c1[1] + p2_c2[1]))
77      newPoints = pointStats.map(
78          lambda st: (st[0], st[1][0] / st[1][1])).collect()
79
80      tempDist = sum(np.sum((kPoints[iK] - p) ** 2) for (iK, p) in newPoints)
81
82      for (iK, p) in newPoints:
83          kPoints[iK] = p
84
85      iteration-=1
```

Run the Script

First upload the data files into HDFS

```
hdfs dfs -mkdir -p /user
hdfs dfs -mkdir -p /user/${your_user_name}
hdfs dfs -put ./data /user/${your_user_name}/

# example:
# hdfs dfs -mkdir -p /user
# hdfs dfs -mkdir -p /user/sakkas
# hdfs dfs -put ./data /user/sakkas
```

Open the `run.sh` file and change username with your user name

```
1 #!/bin/bash
2
3 numExecutor=48
4 coresPerWorker=1
5 memExecutor=4G

6 # change username with your username
7 data=/user/username/data/kmeans-P200000-D10.txt
8 # example:
9 # data=/user/sakkas/data/kmeans-P200000-D10.txt
10 centroids=10
11 iteration=10
```

```
12
13 spark-submit --deploy-mode client --num-executors ${numExecutor} --executor-
cores ${coresPerWorker} --executor-memory ${memExecutor} ./kmeans.py ${data}
${centroids} ${iteration}
```

The parameters of `spark-submit` are as follows

- deploy-mode: use client in our case
- num-executors: the number of executors (parallel computation worker)
- executors-cores: the physical core used by each executor
- executor-memory: the memory usage of each executor

We use 48 executors to occupy all of the juliet nodes cores (48 cores), and set up 4GB memory per executor.

By running the scripts

```
./run.sh
```

The spark shall start and print out the log content to the screen

```
2019-03-22 22:50:25 INFO  RMProxy:98 - Connecting to ResourceManager at j-
029/172.16.0.29:8132
...
2019-03-22 22:50:38 INFO  Client:54 - Application report for
application_1553307084028_0011 (state: RUNNING)
...
2019-03-22 22:50:38 INFO  YarnClientSchedulerBackend:54 - Application
application_1553307084028_0011 has started running.
2019-03-22 22:50:38 INFO  Utils:54 - Successfully started service
'org.apache.spark.network.netty.NettyBlockTransferService' on port 45075.
2019-03-22 22:50:38 INFO  NettyBlockTransferService:54 - Server created on j-
029.juliet.futuresystems.org:45075
2019-03-22 22:50:38 INFO  BlockManager:54 - Using
org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
2019-03-22 22:50:38 INFO  BlockManagerMaster:54 - Registering BlockManager
BlockManagerId(driver, j-029.juliet.futuresystems.org, 45075, None)
2019-03-22 22:50:38 INFO  BlockManagerMasterEndpoint:54 - Registering block manager
j-029.juliet.futuresystems.org:45075 with 10.5 GB RAM, BlockManagerId(driver, j-
029.juliet.futuresystems.org, 45075, None)
2019-03-22 22:50:38 INFO  BlockManagerMaster:54 - Registered BlockManager
BlockManagerId(driver, j-029.juliet.futuresystems.org, 45075, None)
2019-03-22 22:50:38 INFO  BlockManager:54 - Initialized BlockManager:
BlockManagerId(driver, j-029.juliet.futuresystems.org, 45075, None)
2019-03-22 22:50:38 INFO  JettyUtils:54 - Adding filter
```

```
org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /metrics/json.
2019-03-22 22:50:38 INFO  ContextHandler:781 - Started
o.s.j.s.ServletContextHandler@7acb3227{/metrics/json,null,AVAILABLE,@Spark}
2019-03-22 22:50:38 INFO  EventLoggingListener:54 - Logging events to
file:/scratch_hdd/sakkas/spark/history/application_1553307084028_0011
2019-03-22 22:50:40 INFO  YarnSchedulerBackend$YarnDriverEndpoint:54 - Registered
executor NettyRpcEndpointRef(spark-client://Executor) (172.16.0.5:49370) with ID 1
2019-03-22 22:50:40 INFO  BlockManagerMasterEndpoint:54 - Registering block manager
j-005.juliet.futuresystems.org:36066 with 2004.6 MB RAM, BlockManagerId(1, j-
005.juliet.futuresystems.org, 36066, None)
...
```

We saw that the resource manager is registered to the namenode (e.g., j-029), and the executors are registered across the two slave nodes (e.g., j-005 and j-029) with executor ID.

```
The final results shall be similar to this
Final centers: [array([-0.382, -0.322,  0.418, -0.091, -0.431, -0.297,  0.383,
-0.395,
       -0.281, -0.428]), array([ 0.454,  0.135,  0.451, -0.397, -0.291, -0.214,
-0.476, -0.386,
        0.066,  0.367]), array([-0.349, -0.297,  0.525, -0.054,  0.488, -0.164,
0.337,  0.22 ,
        0.521,  0.141]), array([ 0.26 , -0.473, -0.032, -0.551,  0.279, -0.049,
-0.296,  0.007,
       -0.514, -0.436]), array([-0.186,  0.272, -0.248, -0.164,  0.394, -0.323,
0.489, -0.535,
       -0.306,  0.421]), array([ 0.32 , -0.545,  0.189,  0.383, -0.099,  0.362,
0.316, -0.024,
       -0.394,  0.473]), array([ 0.443, -0.25 , -0.273, -0.168,  0.374,  0.375,
-0.279, -0.267,
        0.546, -0.398]), array([ 0.56 ,  0.111,  0.207,  0.512,  0.317, -0.548,
0.029,  0.017,
       -0.149, -0.371]), array([-0.191, -0.312, -0.445,  0.072,  0.42 , -0.436,
-0.515,  0.132,
       -0.037,  0.496]), array([-0.365,  0.353,  0.331,  0.52 , -0.489, -0.163,
-0.307,  0.294,
       -0.078,  0.388]), array([ 0.364,  0.449, -0.489,  0.135, -0.461,  0.356,
0.072, -0.423,
       -0.346, -0.059]), array([-0.5  , -0.267, -0.143, -0.357, -0.349,  0.485,
-0.118, -0.312,
        0.358,  0.383]), array([-0.23 ,  0.364,  0.336, -0.531, -0.055, -0.117,
0.029,  0.53 ,
       -0.517,  0.256]), array([-0.309, -0.44 , -0.464,  0.47 , -0.381,  0.028,
-0.175,  0.393,
       -0.044, -0.412]), array([ 0.424, -0.224, -0.38 , -0.228, -0.408, -0.484,
```

```
0.406,  0.235,
        0.369,  0.219]), array([ 0.384,  0.432, -0.176,  0.158,  0.402,  0.428,
-0.006,  0.393,
        0.29 ,  0.506]), array([-0.368,  0.32 , -0.502, -0.324,  0.279,  0.182,
0.39 ,  0.424,
       -0.112, -0.449]), array([-0.301,  0.156,  0.352,  0.389,  0.497,  0.517,
-0.239, -0.211,
       -0.399, -0.274]), array([ 0.326,  0.173,  0.491, -0.051, -0.427,  0.405,
0.191,  0.328,
        0.386, -0.475]), array([-0.425,  0.453, -0.168,  0.202,  0.009, -0.361,
-0.226, -0.418,
        0.512, -0.391])]
Computing time 34.503 secs
```