Tutorials 2 and 3

Content

- Spark, pyspark, mesos, yarn

Read to the documentation below as required (note that there exist various Spark versions).
http://spark.apache.org/docs/latest/
http://spark.apache.org/docs/0.8.1/
http://spark.apache.org/docs/0.8.1/python-programming-guide.html
http://spark.apache.org/docs/0.8.1/scala-programming-guide.html#master-urls
https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html
https://spark.apache.org/docs/2.3.2/quick-start.html

Find all files (words.txt, file1, file2, and file3) required for this tutorial here:
https://www.cl.cam.ac.uk/teaching/1819/CloudComp/materials.html

Spark

Step 1: Pyspark, Basic Commands

Install the latest version of Spark (2.3.2) on your local machine in Standalone mode. Before that, you also need to install Java, Python and Scala. Run pyspark and using interactive commands run the following queries (practice to work with basic pyspark commands such as map, reduce, reduceByKey, sortByKey, groupBy, groupByKey, filter, mapValues, etc): (10 min)

1. *List the 5 first words (in ascending order) and from the words.txt which start with "b" and end with "t".*
2. *List the 10 last longest words from the file words.txt.*
3. *Calculate the number of lines and the number of distinct words from file1.*
4. *Find 3 common words in files1, file2 and file3 which their sum of frequencies (number of occurrences) within the three files is maximum compared to the other shred words . (e.g., assume that X is a set of words which are shared between all files, find common words like xi in X which maximise F(xi)= f1(count xi in file 1) +f2(count xi in file 2) +f3(count xi in file 3).*
5. *Group words for words.txt according to their first 4 characters and then output the number of members for the first 10 groups.*

Step 2: Working with Spark Standalone Cluster

Write a python script using pyspark to list the 10 most frequent words in file1 in descending order (based on frequency). Ignore all punctuations and do not process any of your customised set of stopwords (e.g., and,or,that,the,a,an,is,are,have). Run your program on your local Spark using spark-submit. While the application runs, monitor running tasks, executers and storage

usage by accessing the driver's UI at http://localhost:4040 . Change the RDD persist level for your Spark application (to MEMORY_ONLY and DISK_ONLY), rerun your application and compare the execution time for both MEMORY_ONLY and DISK_ONLY.

Step 3: Spark on Mesos/Yarn

Spark can be deployed using different cluster managers including its own Spark Standalone, Apache Mesos, Hadoop Yarn and Kubernetes. In steps 1 and 2 you worked with Spark's Spark Standalone cluster manager to allocate resources to applications. Now, try using other alternative cluster managers. Create a Mesos/Yarn cluster, configure the connection between Spark and the external cluster manager, and run the application you developed for the Step 2 on Spark. For faster setup you can try using Docker images. You can also change existing images and push them as new images.

Step 4: Dynamic Changes

For a Kubernetes cluster, how can you dynamically add/remove nodes during application runtime? How can you ask the kube-scheduler to stop/resume scheduling new pods on certain nodes? Which information are used by the scheduler in order to make scheduling decisions (i.e., assigning nodes to pods)? How your-custom application, as a pod, can access similar information?s

See the documentations for the kubernetes python/java client api
https://kubernetes.io/docs/reference/using-api/client-libraries/
https://github.com/kubernetes-client/java/
https://github.com/kubernetes-client/python/